SPECIFICATION, CONFIGURATION AND EXECUTION
OF DATA-INTENSIVE SCIENTIFIC APPLICATIONS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the
Graduate School of The Ohio State University

By

Vijay S. Kumar, B.E. (Hons.), M.S.
Graduate Program In Computer Science and Engineering

The Ohio State University

2010

Dissertation Committee:

Prof. P. Sadayappan, Advisor
Prof. Joel H. Saltz
Prof. Gagan Agrawal
Prof. Umit V. Catalyurek
© Copyright by

Vijay S. Kumar

2010
ABSTRACT

Recent advances in digital sensor technology and numerical simulations of real-world phenomena are resulting in the acquisition of unprecedented amounts of raw digital data. Terms like ‘data explosion’ and ‘data tsunami’ have come to describe the uncontrolled rate at which scientific datasets are generated by automated sources ranging from digital microscopes and telescopes to in-silico models simulating the complex dynamics of physical and biological processes. Scientists in various domains now have secure, affordable access to petabyte-scale observational data gathered over time, the analysis of which, is crucial to scientific discoveries and furthering of knowledge within the domain. The availability of commodity components have fostered the development of large distributed systems with high-performance computing resources to support the execution requirements of scientific data analysis applications. Increased levels of middleware support over the years have aimed to provide high scalability of application execution on these systems. However, the high-resolution, multi-dimensional nature of scientific datasets, and the complexity of analysis requirements present challenges to efficient application execution on such systems. Traditional brute-force analysis techniques to extract useful information from scientific datasets may no longer meet desired performance levels at extreme data scales.

This thesis builds on a comprehensive study involving multi-dimensional data analysis applications at large data scales, and identifies a set of advanced factors or
parameters to this class of applications which can be exploited in domain-specific ways to obtain substantial improvements in performance. Factors like the on-disk layout of datasets and the mechanisms for accessing them, and the mapping of analysis processes to computational resources can be customized for performance based on our knowledge of an application’s computational and I/O properties. A useful property of these applications is their ability to operate at multiple performance levels based on a set of trade-off parameters, while providing different levels of quality-of-service (QoS) specific to the application instance. To avail the performance benefits brought about by such factors, applications must be configured for execution in specific ways for specific systems. Middleware support for such domain-specific configuration is limited, and there is typically no integration across middleware layers to this end. Low-level manual configuration of applications within a large space of solutions is error-prone and tedious.

This thesis proposes an approach for the development and execution of large scientific multi-dimensional data analysis applications that takes multiple performance parameters into account and supports the notion of domain-specific configuration-as-a-service. My research identifies various aspects that go into the creation of a framework for user-guided, system-directed performance optimizations for such applications. The framework seeks to achieve this goal by integrating software modules that (i) provide a unified, homogeneous model for the high-level specification of any conceptual knowledge that may be used to configure applications within a domain, (ii) perform application configuration in response to user directives, i.e., use the specifications to translate high-level requirements into low-level execution plans optimized for a given system, and (iii) carry out the execution plans on the underlying system
in an efficient and scalable manner. A prototype implementation of the framework that integrates several middleware layers is used for evaluating our approach. Experimental results gathered for real-world application scenarios from the domains of astronomy and biomedical imaging demonstrate the utility of our framework towards meeting the scientific performance requirements at very large data scales.
To my parents, Bharathy and Sundaram Shiv Kumar,

Ashwin, family and friends.
ACKNOWLEDGMENTS

I would like to acknowledge my dissertation committee consisting of Dr. P Sadayappan, Dr. Joel Saltz, Dr. Gagan Agrawal, and Dr. Umit Catalyurek. I am grateful to my advisor, Dr. Saltz, for his continued support and guidance throughout my graduate study. I would like to thank Dr. Sadayappan for taking on an advisory role to allow me to continue pursuing my thesis while working on a remote basis. I am especially grateful to Dr. Tahsin Kurc for closely mentoring and monitoring my work over the years. The high levels of energy and constructive criticism he offered in our discussions have always spurred me to give my best efforts. Thanks to Benjamin Rutt for showing me the high standards that any experimental systems developer ought to strive for. I would also like to acknowledge my research collaborators including Ghaleb Abdulla and Yolanda Gil for their words of advice.

Graduate school has been a life-changing experience for me, and I am thankful to my close friends who eased me through this transition. K2, thanks for being a very supportive lab-mate – I always enjoyed our interactions, both technical and otherwise. I owe an enormous debt of gratitude to my close friends from graduate school: Gaurav, Sitaram, Abhilash, Yashaswini, and Miti for some of life’s better moments experienced over the last seven years. I am very thankful to Sudha for her patience – thanks for constantly giving me reason to believe in a life beyond my thesis. To other friends whose company I have immensely enjoyed and always look
forward to – Rajkiran, Tyagi, Muthu, Shirish, Shantanu and Danjy – a big thanks to all of you.

Last but by no means the least, I would especially like to acknowledge my parents, Mr. and Mrs. Kumar, for their constant support and encouragement in the face of so many troubles and sacrifices endured over the years. There were numerous occasions when I felt this honest emptiness for being away from home when there were obvious needs to be by your side. Thanks for being so considerate and understanding and for never suggesting a shortcut to success. You’ve taught me that an excellent education is paramount, and I will strive to make the most of this opportunity all through my life. To my brother Ashwin, it was unfortunate we never got the chance to live together while you were at OSU, but thanks for understanding that it was in the interests of a better future. Your presence and positive outlook to life did give me a lot of strength during the difficult times. Finally, thanks to my very supportive extended family for all their blessings, love and affection.
VITA

November 23, 1980 .......................... Born - Chennai, India

1998 – 2003 ................................. B.E. (Hons.) Computer Science +
M.Sc. (Hons.) Chemistry
Birla Institute of Technology and Science (BITS),
Pilani, India

Jan 2003 - Jun 2003 .......................... Intern,
i2 Technologies (India) Pvt. Ltd.,
Bangalore, India

2003 – 2010 ................................. Graduate Research Associate,
The Ohio State University.

Jul 2007 - Sep 2007 .......................... Student Research Scholar,
Institute for Scientific Computing Research,
Lawrence Livermore National Laboratory.

PUBLICATIONS


**FIELDS OF STUDY**

Major Field: Computer Science and Engineering
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Dedication</td>
<td>v</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>vi</td>
</tr>
<tr>
<td>Vita</td>
<td>viii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xiv</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xv</td>
</tr>
<tr>
<td>Chapters:</td>
<td></td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Motivation</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Research Issues and Contributions</td>
<td>10</td>
</tr>
<tr>
<td>1.4 Organization of the Thesis</td>
<td>25</td>
</tr>
<tr>
<td>2. Motivating Applications and Science Requirements</td>
<td>26</td>
</tr>
<tr>
<td>2.1 Application Motifs: A case for conceptual models</td>
<td>26</td>
</tr>
<tr>
<td>2.2 Target Application Domains</td>
<td>29</td>
</tr>
<tr>
<td>2.2.1 Biomedical Imaging</td>
<td>29</td>
</tr>
<tr>
<td>2.2.2 Astronomy Sky Surveys</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3 Particle Physics</td>
<td>40</td>
</tr>
<tr>
<td>2.2.4 Satellite Imagery</td>
<td>42</td>
</tr>
<tr>
<td>2.3 Characterizing Application Workloads</td>
<td>43</td>
</tr>
<tr>
<td>2.4 Summary</td>
<td>51</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Data generation rates within some scientific domains</td>
<td>2</td>
</tr>
<tr>
<td>4.1 Query Performance results</td>
<td>121</td>
</tr>
<tr>
<td>4.2 LSST Crossmatch Execution on Parallel Database System</td>
<td>142</td>
</tr>
<tr>
<td>4.3 LSST Crossmatch execution on a Network Database System</td>
<td>143</td>
</tr>
<tr>
<td>4.4 Crossmatch time using 16x1 strategy (OptZones algorithm), FOVObject vs. DIASources</td>
<td>146</td>
</tr>
<tr>
<td>5.1 Performance and accuracy at varying image resolutions.</td>
<td>169</td>
</tr>
<tr>
<td>5.2 Dataset characteristics</td>
<td>181</td>
</tr>
<tr>
<td>5.3 Influence of image content at threshold 0.7</td>
<td>183</td>
</tr>
<tr>
<td>7.1 Number of tasks in PIQ workflow instance for a given chunksize parameter value</td>
<td>236</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Middleware support for large-scale data analysis applications</td>
<td>4</td>
</tr>
<tr>
<td>1.2 The Montage Workflow designed by scientists from the NASA/IPAC Infrared Science Archive for astronomers to generate custom mosaics from multiple sky images</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Stages in a typical workflow for large, scientific multi-dimensional data analysis</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Large-scale structure in a standard cold-dark matter Universe. Courtesy R. Cen, Princeton University</td>
<td>8</td>
</tr>
<tr>
<td>1.5 Hubble’s Tuning Fork classification of galaxies borrowed from Hubblesite</td>
<td>16</td>
</tr>
<tr>
<td>1.6 Galaxy classification based on non-morphological features</td>
<td>17</td>
</tr>
<tr>
<td>1.7 Design of framework to support performance optimizations for large scientific multi-dimensional data analysis</td>
<td>24</td>
</tr>
<tr>
<td>2.1 Grid showing twelve application motifs identified from computational patterns within different application domains. Color of a cell represents the relative presence of the pattern within the domain – red(high), blue(low). Courtesy of Berkeley Par Lab</td>
<td>27</td>
</tr>
<tr>
<td>2.2 Formulation of aggregation query as a relational database query</td>
<td>32</td>
</tr>
<tr>
<td>2.3 The Pixel Intensity Quantification (PIQ) application</td>
<td>33</td>
</tr>
<tr>
<td>2.4 Data analysis workflow for studying Phenotypical change</td>
<td>36</td>
</tr>
</tbody>
</table>
3.13 XIO Cluster Collective End-to-End Preprocessing Rate (16 nodes, 6GB files, local configuration) ................................................................. 102
3.14 Summary of contributions in this chapter ................................................................. 103
4.1 Overall time taken: Local v/s Remote component placement for PIQ transformation tasks (5 GB image, 16 nodes overall) .................................................. 108
4.2 Percentage of time spent in computation by the PIQ transformation tasks (5 GB image, 16 nodes overall) ................................................................. 109
4.3 Workflow mapping with optimal configuration for individual components (256 GB image, 32 nodes overall) ................................................................. 110
4.4 User-directed component placement that also considers data transfer 113
4.5 Full and Local Prefix Sum Generation, 12 GB Dataset, 16 Nodes . . . 118
4.6 Computation of full prefix sum using local prefix sums (with 16 total chunks) ................................................................. 119
4.7 Algorithm to compute full prefix sum at a point using local prefix sums 120
4.8 Warping of a brain mosaic to a standard brain atlas ......................... 126
4.9 ODM configuration for parallel 2D inverse image warping ................. 131
4.10 Effects of large chunk size leading to increased memory requirements of ODM ................................................................. 132
4.11 Warp component performance under three different configurations . 135
4.12 Spatial parameterization of the sky and the use of the zones index . 139
4.13 SQL query for the Zones crossmatch algorithm (DIASource vs. FOVObject) 139
4.14 SQL crossmatch query for zone i in the OptZones algorithm ............. 140
4.15 Using concurrency to resolve mismatch on MySQL Cluster ............... 144
4.16 Varying replication factors and query partitioning mechanisms (Opt-Zones algorithm, high density FOV region, DIASource vs. FOVObject) 148

4.17 Summary of contributions in this chapter (highlighted in red) 150

5.1 Data analysis workflow for Neuroblastoma classification 166

5.2 Basic strategy: No parameter configuration (RAND) 171

5.3 Simulation studies for accuracy parameter configuration. 100 tasks, 10 resolutions per task 172

5.4 Simulation studies when accuracy performance characteristics are not linearly proportionate 173

5.5 The QueueEntry class 175

5.6 Object-based heuristic is not useful for application-level QoS requirements at the granularity of chunks or regions (Neuroblastoma classification) 177

5.7 Console filter scheduler algorithm (Region-based heuristic) 180

5.8 Benefits of region-based heuristic for application-level QoS: “Maximize number of finalized regions in time \( t \)” 182

5.9 Benefits of region-based heuristic for application-level QoS: “Maximize average confidence across all regions in time \( t \)” 184

5.10 Percentage increase in average confidence with varying confidence thresholds, 32 workers 185

5.11 Scalability tests for accuracy trade-off heuristics 186

5.12 Summary of contributions in this chapter (highlighted in red) 188

6.1 Specification and Configuration within an application instance 194

6.2 Wings data ontology extensions for spatial, multidimensional data 221

7.1 IMPACT Framework architecture 224
7.2 PIQ application workflow represented using Wings .................. 226
7.3 Meta-components in the Neuroblastoma Classification workflow instance 229
7.4 Effect of varying chunksize parameter on PIQ workflow execution . . 238
7.5 Wings workflow template for the PIQ application with meta-components 240
7.6 Execution time with different task granularity (5GB image, 40 nodes, chunksize=512 × 480) ......................................................... 241
7.7 Autotuning to obtain optimal value of numActiveChunks parameter for the warp component; 8 processors ................................. 244
7.8 Using accuracy parameter tuning to improve response under time constraints ................................................................. 246
7.9 Quality and scalability tests for requirement 2 ......................... 248
CHAPTER 1

INTRODUCTION

Data analysis applications in many domains are witnessing a steadily growing trend in their complexity, brought about by the larger volumes of raw data to be analyzed, and the increasing complexity of numerical models employed to study real-world phenomena. From astronomy to biomedical imaging and pathology to molecular dynamics, large-scale simulations and advanced digital sensor instruments are equipped to generate up to petabyte-scale multi-dimensional datasets over time. Data generation rates for instances of large-scale undertakings within the earth sciences, physical sciences and life sciences are shown in Table 1.1. The raw datasets in each case essentially consist of a large number of data points that correspond to measurements of attributes over some multi-dimensional space used to model the phenomena under observation. For instance, in climate modeling studies (e.g., “What are the effects of global warming on the vegetation of the Amazon rainforest?”), raw data may contain attributes such as temperature, pressure, relative humidity, dew point, and other measurements recorded at different points and elevations on relevant regions of the earth’s surface over time.

Recent trends point to a growth in such dataset volumes at a rate that far outpaces Moore’s law for growth in computing power [101, 102, 61, 53]. Scientists are
<table>
<thead>
<tr>
<th>Scientific Domain</th>
<th>Typical Science Runs</th>
<th>Data Volumes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fusion/Plasma Simulation</td>
<td>1 billion particles</td>
<td>200 GB/minute</td>
</tr>
<tr>
<td></td>
<td>125 million grid points</td>
<td></td>
</tr>
<tr>
<td>Climate Modeling</td>
<td>6 million grid points</td>
<td>8 PB per simulation</td>
</tr>
<tr>
<td></td>
<td>100 year simulation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\approx 100 variables/grid point</td>
<td></td>
</tr>
<tr>
<td>Nuclear Reactor Simulation</td>
<td>120 million elements</td>
<td>900 TB - 1.2 PB per run</td>
</tr>
<tr>
<td></td>
<td>7 variables per element</td>
<td></td>
</tr>
<tr>
<td></td>
<td>90k timesteps</td>
<td></td>
</tr>
<tr>
<td>Astronomy Sky Surveys</td>
<td>Map entire southern sky</td>
<td>12-15 TB/night</td>
</tr>
<tr>
<td></td>
<td>3.2 Gigapixel image</td>
<td>Total 55 PB in 10 years</td>
</tr>
<tr>
<td></td>
<td>every 15 seconds</td>
<td></td>
</tr>
<tr>
<td>Digital Pathology</td>
<td>Scan tissue samples</td>
<td>3-5 GB/minute</td>
</tr>
<tr>
<td></td>
<td>\geq 150k \times 150k pixel images</td>
<td>(per microscope)</td>
</tr>
<tr>
<td></td>
<td>Several focal planes</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.1: Data generation rates within some scientific domains

turning to automated services and other software technologies in a bid to cope with ever higher volumes of data. They rely extensively on high-performance distributed computing systems built from commodity components to affordably analyze these large datasets to arrive at hypotheses. The analysis applications that reduce datasets into comprehensible information must meet desirable performance levels regardless of the underlying compute infrastructure for them to be useful. Hence, in the face of increasing data volumes and generation rates, the problem of determining suitable execution strategies from a performance point of view, and their realization in software is an important one. The focus of my research is on investigating strategies for efficient execution of complex data analysis processes over very large multi-dimensional datasets characterized by high spatial and temporal resolutions. The insights gained from my research are used to lay the groundwork for high-level specification and configuration mechanisms for analysis services that are
aimed at easing the development of such applications for execution on large-scale high-performance computing (HPC) systems.

1.1 Background

Over the past decade, traditional cluster systems built using commodity components emerged as the popular platform for carrying out large-scale parallel computations over massive scientific data. Grid-based infrastructures went a step further to allow sharing of compute resources and data over the wide area, thereby providing highly scalable and transparent HPC support to end-users. That is, vast amounts of computing power could be provisioned from geographically disparate sources (or Grid sites) including high-end clusters and idle workstations, but the details are hidden from the user. More recently, the Cloud model [15] built out of large data centers housing modern heterogeneous cluster systems, and consisting of virtualized resources equipped with hardware acceleration are finding greater use among scientists for parallel computations [93, 3]. Advances in middleware tools and techniques have enabled “big data” analysis applications to harness the compute resources provided by clusters and web-/Grid-scale systems. The data-intensive computing toolkit at the disposal of the users ranges from lower-level process-centric software to higher-level data-centric software layered upon these lower levels (Figure 1.1). These middleware layers are responsible for complex process- and data-management related operations ranging from orchestrating the execution of a process on a node within the system, to ensuring reliable execution, to transferring data between nodes, to name a few.

Data analysis applications have also transitioned from monolithic pieces of code to collections of interdependent code modules. Applications are typically expressed
### Data-Intensive Analysis Applications

<table>
<thead>
<tr>
<th>Applications</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwiftScript, Wings, VDL, Scufl, Pig, DryadLINQ, SPL</td>
<td>Application-centric middleware for high-level representations of application structure and requirements</td>
</tr>
<tr>
<td>Karma, VisTrails, PReServ, STAMPEDE</td>
<td>Data-centric middleware for data-driven processes</td>
</tr>
<tr>
<td>DAGMan, Falkon, Pegasus, Swift, Kepler, Taverna, Trident</td>
<td>Process-centric middleware for managing individual clusters</td>
</tr>
<tr>
<td><strong>Workflow Management Systems:</strong></td>
<td></td>
</tr>
<tr>
<td>Services/Task-based, Abstract/Concrete</td>
<td></td>
</tr>
<tr>
<td>(Workflow execution &amp; monitoring, High-level language, Provenance support)</td>
<td></td>
</tr>
<tr>
<td><strong>Scalable Distributed Dataflow Processing</strong></td>
<td></td>
</tr>
<tr>
<td>(Object-oriented/Component Frameworks, Language and Runtime Support)</td>
<td></td>
</tr>
<tr>
<td><strong>OSCAR, Rocks, Scali, Hadoop:</strong></td>
<td>Cluster-level Process Management</td>
</tr>
<tr>
<td>(Batch systems, Job scheduling/Resource Monitoring, Parallel Programming)</td>
<td></td>
</tr>
<tr>
<td><strong>Resouces:</strong></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>(interconnect), storage, CPUs/GPUs within each cluster</td>
</tr>
<tr>
<td><strong>Protocols &amp; specifications for sharing of resources,</strong></td>
<td></td>
</tr>
<tr>
<td><strong>services and data on the Grid/Cloud platforms</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Condor-G, TeraGrid, OSG</strong></td>
<td></td>
</tr>
<tr>
<td>AWS, Azure, App Engine</td>
<td></td>
</tr>
<tr>
<td><strong>Globus Toolkit (GSI, GridFTP, GRAM, RLS)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SOA (SOAP, WSDL, UDDI)</strong></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1.1: Middleware support for large-scale data analysis applications
in the form of dataflow graphs or workflows or networks of analysis tasks with well-defined data dependencies between the tasks. That is, complex analysis applications are composed from a large number of smaller tasks that solve parts of the bigger scientific problem. Workflows themselves can be defined recursively as being composed of a number of smaller workflows. Individual analysis tasks, also known as workflow components, may correspond to legacy scientific codes (or services in a service-oriented architecture (SOA)). Each component corresponds to a user-defined analysis operation. Each such operation takes one or more input data instances and processes them in user-defined ways to produce one or more output data instances. Multiple users could develop the codes corresponding to different components within the same workflow. Figure 1.2 shows an instance of the well-known Montage [7] application workflow borrowed from the astronomy domain. Here, raw image data obtained from telescopes are processed through a sequence of high-level components such as projection, filtering and background correction. Some of these operations could be common across multiple application domains outside of astronomy. Each instance of a component performs similar operations on different data instances. Within different domains, one can observe specialized algorithms for each operation.

1.2 Motivation

My thesis work is motivated by the scientists’ need to analyze large, high-resolution scientific multi-dimensional datasets so as to extract useful information. To understand the motivations, it is important to understand the stages involved in carrying out analysis over such datasets (Figure 1.3). The first stage involves the generation or

1In the remainder of the text, the use of the term component instance could be considered analogous to a service in SOA
Figure 1.2: The Montage Workflow designed by scientists from the NASA/IPAC Infrared Science Archive for astronomers to generate custom mosaics from multiple sky images.

Figure 1.3: Stages in a typical workflow for large, scientific multi-dimensional data analysis
acquisition of large-scale datasets from various sources including simulations and sensors. This raw data is typically archived for future read-only access. However, prior to long-term storage, there are intermediate stages where scientists require several forms of analysis tasks to be performed on the data as it streams in from the source. These tasks could be categorized into low-level data processing tasks and high-level data analysis tasks. Typically, low-level tasks operate on raw data to reduce them to one or more intermediate data forms as required by the downstream tasks; in turn, the high-level tasks operate on the intermediate data to identify domain-specific data objects, features, and other interesting regions within the datasets, the study of which may provide the required insight and knowledge that the scientists are looking for.

Scientific data analysis is often iterative and interactive – although the workflows are characterized by a Directed Acyclic Graph (DAG) based structure, execution of parts of the workflow may need to be carried out multiple times until the user obtains the desired results. As a rule of thumb, the further one navigates into such analysis workflows away from the source, the lesser the data volumes to be analyzed, while the complexity and domain-specificity of the analysis tasks increases.

Using an example from the astronomy domain, these stages can be explained. Deep sky exploration initiatives, such as the Large Synoptic Survey Telescope [6] and PanSTARRS [8] involve the continuous capture of large, high-resolution images corresponding to regions of the night sky, possibly at multiple wavelengths and focal planes. Figure 1.4 shows such an image corresponding to a 4 sq. degree region of the sky. In this case, objects of interest within the image visually manifest themselves as light sources against a dark background. A scientist may select a region of interest within the sky and seek to study the existence (or lack thereof) of dark matter and
Figure 1.4: Large-scale structure in a standard cold-dark matter Universe. Courtesy R. Cen, Princeton University.
other sub-stellar objects such as brown dwarfs and quasars that are believed to lie at the deep end of the universe. For this, the image data first needs to undergo a series of data correction tasks aimed at eliminating any artefacts and errors that may have been introduced on account of improper calibration and movement of the sensor instrument during image capture. All images that fall within the region of interest must be stitched together on-demand to form a seamless mosaic – this intermediate data could then be used for comparative analysis against data obtained from alternative sources. The steps described so far constitute the low-level processing tasks that operate on the raw image (pixel) data. For instance, the scientist could use the Montage workflow described in section 1.1 for mosaic creation. High-level analysis tasks such as segmentation, feature extraction and several forms of classification may then be employed to identify objects of interest (such as brown dwarfs) within the image mosaic.  

The scientist may then perform iterative analysis to refine the objects of interest and study their properties further. Results obtained through comparative analysis against other such studies, quantification and insights gained through visualization of the reduced data could potentially lead to new scientific discoveries and knowledge about dark matter. Applications in other scientific domains can also be considered to be analogous to the above example in terms of how they go about transforming raw multi-dimensional datasets into useful information.

The fundamental capability of processing large multi-dimensional data volumes notwithstanding, scientists are also challenged by the need to efficiently carry out complex computations so that analysis can keep up with the raw data generation

\[2\] An image mosaic is an instance of 2D spatial data. In general, scientific applications may require similar analysis to be performed over higher dimensional datasets.
rates. The lack of simple generalizability and the inherent complexity of analysis applications coupled with the heterogeneity in hardware and software architectures of today’s systems are inter-related factors which, if not well-understood, could lead to severe performance bottlenecks during execution. This makes the efficient and scalable execution of applications a difficult job for the scientists. My thesis is motivated by some of the challenges presented by such applications:

1. **Large data scales**: Low-level processing tasks constitute the initial part of most application workflows. Raw multi-dimensional datasets are transformed into intermediate data products through these tasks. The common aspect to these tasks is that they typically view input datasets as a collection of **data points** in some \( n \)-dimensional data space. Each data point is a zero-dimensional spatial entity (e.g., a pixel or an ion) in that it has no spatial extent and is just a point in space that is associated with a vector of spatial coordinates, e.g., \( (x, y, z) \) for a 3D Euclidean space or \( (ra, decl) \) for the equatorial coordinate system. However, each data point is also usually associated with a unique identifier and a relatively small albeit increasing number (\( \approx 10 \) to \( 20 \)) of non-spatial attributes that correspond to measurements recorded or simulated at that point in space. Examples of such attributes could include **momentum** and **weight** for molecular dynamics simulations, **wavelength band** for hyperspectral imaging, **color intensity values** in different color coding schemes for microscopy image data, and **oil saturation** and **pressure** for oil-well placement simulation studies.

When such low-level tasks are invoked to process terabytes or petabytes of high-resolution data over time, the limited amounts of physical memory available on the systems renders the staging of entire datasets infeasible, and the reliance on virtual
memory heavily impacts performance negatively; Raw datasets and any intermediate data must therefore be partitioned and stored across disks on one or more storage nodes within the cluster. As a result, it becomes impractical for workflow tasks to assume that all the data they need can fit within memory at once. Tasks must be modified to operate out-of-core, i.e., to access only relevant portions of distributed disk-resident datasets on-demand. Depending on the complexity of the transformation function, the tasks may require repeated access over large portions of the data. When the input and/or output multi-dimensional data spaces contain billions to trillions of data points, repeated accesses to disk could contribute to significant I/O overheads in the overall execution time. Expecting end-users to modify their codes to account for this distributed I/O is not favorable as it would require them to be cognizant of analysis tasks developed by other users. Moreover, without user awareness of the underlying resources, it would be impossible to provide efficient I/O. These drawbacks point to an approach where analysis tasks must be augmented with a layer of abstract **multi-dimensional data services** that can perform any I/O-related activities on behalf of the users. The challenges from a software support perspective include providing mechanisms for (i) transparently serving the data management needs of all forms of analysis tasks by providing them a layer of storage and I/O virtualization for cluster environments, (ii) aptly configuring the multi-dimensional data services depending on the underlying system architectures, (iii) gracefully scaling to handle very large datasets while placing minimal constraints on the available physical memory, and (iv) customizing the services for runtime performance optimizations within specific domains – e.g., data could be partitioned and laid out on disk in application-specific ways to minimize disk accesses for that application instance. In
other words, we need to provide scientists with a system to minimize execution time of their applications by optimizing the consumption of disk and network bandwidths, and compute resources.

2. Time and Resource Constraints: Due to the high data generation rates, analysis tasks are required to meet stringent constraints on their execution times to keep up with the incoming data. Reductions in execution time achieved by scaling application execution to a large number of compute resources within the underlying system, and by minimizing all data movement for the applications may not be sufficient to meet these constraints.

As mentioned earlier, our target systems are built using commodity components including processors, disks and switches. Depending on the component assembly, these systems are characterized by differences in their hardware architecture. For example, in Grid and Cloud environments, the need for system balance often leads to heterogeneity in hardware: one computational site could be a cluster of relatively slower dual-processor nodes, each equipped with high bandwidth access to a storage system (e.g., local disks), large amounts of physical memory and connected via a Gigabit Ethernet switch; a second computing site could be a cooperative cluster of nodes with multi-core CPUs and hardware accelerators such as GPUs, interconnected using a high-speed fabric like Infiniband. However, to balance costs, these latter nodes may be equipped with relatively lower memory and storage capacity and with low bandwidth access to storage. Even among computational sites with similar hardware architecture, the process- and data-centric middleware used to manage

3One could argue that frameworks like Hadoop/MapReduce have similar goals. While they specifically seek to optimize data-parallel tasks through scale-out parallelism, we additionally target arbitrary heterogeneous-parallel tasks.
low-level operations for analysis tasks could vary. For example, dataflow processing middleware may use data files as the medium of communication between processes, or alternatively use streams (Unix/TCP pipes) or shared-memory queues as communication channels. Batch systems may support opportunistic scheduling for many-task computing (MTC) [92] on idle workstations, and alternatively support dedicated scheduling for time-critical parallel tasks. Such choices characterizing the software architecture within a system could also affect the execution time of applications.

Mapping of workflow analysis tasks onto one or more compute resources in the system, also referred to as workflow scheduling, is an integral part of an application’s lifecycle. To minimize end-to-end execution time for an application and to meet time constraints, scientists seek the mappings that make optimal utilization of resources by its tasks. This mapping could be user-directed or system-directed. In general, mapping DAG-based computational workflows onto multiple resources to minimize execution time, with or without constraints is considered to be NP-complete. Hence, user-directed strategies are unlikely to result in efficient mappings. System-directed mapping often relies on precise analytical performance models for the applications, or on past history of execution. A large number of heuristics have been proposed in the literature addressing system-directed workflow scheduling strategies (with multiple constraints) in heterogeneous environments.

Multi-dimensional data analysis applications present a greater set of challenges with respect to mapping of tasks. Due to multiple data dimensions and the dependence on data content, it is generally impractical to assume the availability of precise analytical performance models for tasks within such applications. Since these applications are characterized by large amounts of intermediate data exchange between tasks,
the workflow scheduling strategies also need to take into account these hard-to-predict data transfer overheads while mapping tasks. Due to the lack of \textit{a priori} information for such applications, system-directed mapping approaches typically use some hints or \textit{knowledge} of the application execution context (e.g., past history of execution) to derive relative performance models that suggest optimal mappings to a certain degree of approximation. With large multi-dimensional data analysis applications, the traditional mapping problem of finding and allocating the optimal set of resources to each task now becomes "\textit{Given a predicted optimal mapping (either user-directed or system-directed) that maps workflow analysis tasks to specific computational resources, how can each task be further configured for performance improvements on its corresponding set of resources in order to improve the end-to-end execution time?}"

The challenges from a software support perspective include the following: (i) Performance of large multi-dimensional data analysis is highly influenced by decisions made with respect to partitioning of computation among the resources, distribution of data, granularity of computation, the computation-communication overlap, and factors such as spatial locality of the data \cite{70}. We need mechanisms to express such high-level knowledge about the application execution context so that system-directed approaches may utilize them to derive more optimal mapping strategies, (ii) As a result of the mapping strategy chosen, an analysis task may be constrained to execute on a specific set of resources. Since multi-dimensional data analysis tasks can be configured for execution on a given system architecture in a variety of ways, there is a potential for existence of several \textbf{mismatches between the requirements of the analysis and the architectural underpinnings of the system}. Examples of such mismatches could include the selection of an algorithm for a task that fails
to exploit hardware features of the system, or, an inefficient implementation of the selected algorithm by the middleware layers available on the system. Mechanisms for identification of such mismatches at a high-level are needed so that system-directed mapping approaches can eliminate sub-optimal application workflow configurations in a more informed manner. In other words, within a specific domain (such as scientific multi-dimensional data analysis in this case), systems must be able to leverage application and dataset properties to navigate the vast search space of workflow configurations in an efficient manner.

3. Complex analysis models: High-level analysis tasks view data as a collection of domain-specific entities including data objects and features in some \( n \)-dimensional space. Examples of such objects within the running example in this chapter could include quasars and brown dwarfs, or stars and galaxies. Each data object corresponds to some human-perceivable \( k \)-dimensional spatial entity \((k > 1)\) within the domain and has some spatial extent (e.g., line, polygon, sphere). For example, in biomedical imaging, cells and nuclei could be characterized by circular shapes, while lesions could have more of an elliptical shape. These objects are usually associated with a unique identifier and hundreds of dimensions or features such as radius and eccentricity that help characterize their shapes comprehensively. As another example based on the morphological properties of data objects, images of galaxies captured by telescopes can be classified into one or more types – Ellipticals, Normal Spirals, Barred Spirals, Lenticulars and their sub-types – E0, SBA, Sc, . . . using Hubble’s classification scheme as shown in figure 1.5. More complex classification schemes such as the MK system by Yerkes-Morgan use both shapes and additional non-morphological features (such as the spectral types of the prominent stars in the galaxy, the inclination of the galaxy
Figure 1.5: Hubble’s Tuning Fork classification of galaxies borrowed from Hubblesite to the viewing angle, and degree of central concentration) to provide a more accurate classification of the galaxies than the Hubble’s scheme (Figure 1.6).

To accurately model human perception of data objects within any domain, high-level tasks employ a set of highly complex numerical models making them very computationally intensive. However, when applied to hundreds of Gigabytes of object data (millions to billions of objects), these analysis tasks can be highly time-consuming and could require several hours or even days of computation on high-end cluster systems. Given the high computational complexity of these tasks, it may not be possible to meet performance requirements by relying solely on optimization strategies that eliminate performance bottlenecks. When all such strategies fail, one option is to exploit a specific property of these tasks that allows them to operate at multiple performance levels – namely, the forgiving nature of high-level analysis tasks that makes them resilient to errors in computation and data. In other words, since high-level analysis tasks typically have no “perfect” result, it may be possible to sacrifice
(a) Classification of stellar objects based on size (b) Color-magnitude diagram to classify galaxies and color index

Figure 1.6: Galaxy classification based on non-morphological features
the accuracy of such analysis tasks and produce output of lesser quality in order to improve their performance. If a user is tolerant to certain degrees of error in the output, then one can employ lesser complex models or, alternatively, could process only subsets of the input data in an attempt to produce results of lower but acceptable quality in quicker time. For example, in the galaxy classification example shown above, the most accurate classification for the Andromeda galaxy: $kS5$ may take several hours to compute. But, based only on the simple Hubble model, the task may be able to estimate the same result, with upto 80% probability (or confidence in classification) in a matter of minutes, thereby saving huge amounts of execution time and resources. The confidence in classification here is representative of the accuracy of the classification analysis task. A scientist may wish to place bounds on the accuracy so that the system does not seek to produce inordinately low-quality results. For example, the user may state as part of the application requirements that the classification of galaxies be carried out with at least 75% accuracy, in which case the above trade-off would be tolerable to the end-user. Thus, the challenges from a software support perspective include mechanisms to (i) allow various avenues to trade accuracy of multi-dimensional data analysis for performance, (ii) support various notions of application-specific quality-of-service (QoS) within the domain, and provide best-effort guarantees to meet user requirements on the accuracy, and (iii) carry out the trade-offs at various levels in the application workflow in a scalable and automatic manner while operating on large multi-dimensional data.

In summary, the analysis of large scientific, multi-dimensional datasets that are characterized by high spatial and temporal resolutions presents challenges from both an algorithmic and systems point of view. Most of these challenges point to the need
for greater flexibility and **configurability of application execution** so that they can be customized for performance to meet specific user requirements on a given system. In my opinion, this desired level of configurability is provided to a very limited extent by the middleware layers shown in Figure 1.1 and any such support provided across the layers is not integrated. This thesis seeks to provide a single integrated framework that can address the above challenging requirements within the domain of multi-dimensional data analysis applications.

### 1.3 Research Issues and Contributions

The overarching goal of my research is to investigate strategies for improving end-to-end performance of scientific, multi-dimensional data analysis applications on distributed HPC systems at large data scales. To this end, this thesis aims to contribute an integrated framework which allows the of specification of high-level (conceptual) knowledge behind these strategies to assist systems with application configuration.

The research issues addressed in this work are similar in nature to those addressed by High Performance Fortran (HPF) \[57, 58\] compilers for array manipulation, or by query optimizers within the Oracle and MySQL database systems. In HPF, the high-level constructs are extensions to the F90 array language to allow applications to control data partitioning and mapping across multiple processors, and also assist the HPF compilers in translating this high-level specification into an executable program by generating the low-level communication code for a given set of data-parallel operations and data distribution directives. Similarly, in database systems, SQL is extended with high-level hints through which users can direct the query optimizer to use specific indexes or join algorithms based on their understanding of the analysis.
task. This work seeks to provide similar support to enable systems to identify and control execution of large-scale multi-dimensional data analysis applications and meet the desired performance levels. In this section, I explain the specific contributions of this thesis in greater detail:

**Identification of relevant performance parameters.** The first goal of this thesis is to identify relevant *performance parameters* in the context of scientific multi-dimensional data analysis applications, and to study the effects of these parameters on performance as data scales increase to the terabyte and potentially the petabyte ranges. Applications must be configured to generate suitable execution plans for the underlying system architecture. Parameters here correspond to configuration decisions that the user or system makes with respect to different aspects of the application’s execution. To enable analysis applications scale to large data, I first developed a set of multi-dimensional data services via an I/O virtualization or ‘mediator’ layer. Application code can now avail of transparent access to a large underlying storage system (which could be a collection of independent storage nodes that access only local disks, or alternatively, a peer-to-peer storage system with asynchronous replication mechanisms). Besides demonstrating the ability to support several terabytes of data in cluster environments [97, 69], the mediator layer provides advantages over high-performance distributed file systems like Lustre [1] and overlays like STORM [82, 83] by including specific tools and services for storing and repeatedly accessing scientific multi-dimensional data. The mediator layer can incorporate *application-driven data management* customized for a specific domain. Applications can provide high-level directives to the mediator layer (such as the data partitioning scheme to
be used, the layout of data elements on disk and the layout of individual dimensions within these data elements) in order to configure data management operations specific to its requirements.

An instance of a parallel dataflow system was built atop this mediator layer to service the scalability requirements of the large-scale confocal microscopy image analysis domain for use in neuroscience and cancer-related studies – I developed the Out-of-Core Virtual Microscope (OCVM) middleware \cite{97, 69} to provide biomedical image analysis experts a simple, extensible multi-dimensional data model and programming model to allow them to run their analysis codes on their image datasets. OCVM includes customizable services to handle distributed I/O and was used to execute a set of common operations such as image warping on out-of-core image data. Experiments with a set of analysis tasks borrowed from real-world application workflows revealed a set of parameters that could be configured to relieve the performance bottlenecks for large image analysis applications.

The next goal of the thesis was to have multi-dimensional data analysis applications exploit the heterogeneity in architectures of systems composed of multiple clusters or Grid sites connected over the wide area. Such a capability would enable the identification of additional performance parameters related to the mapping of analysis tasks to resources. In order to achieve this goal, I developed extensions to a scalable dataflow middleware system (for multi-dimensional data) like OCVM to integrate it with Grid workflow systems and batch systems \cite{68, 64}. This results in a hierarchical execution module where workflow scheduling across multiple clusters is performed for coarse-grain analysis tasks, and mapping of fine-grain processes (both
analysis and data services) within each analysis task is then performed on a cluster.

**Understanding performance-accuracy trade-offs in multi-dimensional analysis.** An important emphasis of this thesis is on understanding performance-accuracy trade-offs in multi-dimensional data analysis applications, and their role in improving performance and system response to user requirements. In the next work, I studied the common trade-off mechanisms applicable to high-level analysis tasks such as classification using a comprehensive set of simulation experiments. Based on lessons learnt from the simulations, I designed heuristics that carry out appropriate trade-offs in response to different kinds of application-specific quality-of-service (QoS) requirements from the end-user [66, 67]. Some heuristics rely on the specific properties of the analysis tasks, while others rely on the spatial properties of the datasets. The OCVM middleware was extended with additional services that provided the system support to implement the above heuristics at large data scales.

**High-level specification of performance parameters.** Parameters can be categorized into data-level, workflow-level, component-level or system-level based on nature of the configuration decisions. For increasingly complex application requirements and large data volumes, the ‘multi-parameter’ space within which one makes configuration decisions to generate execution plans can grow very large. Moreover, complex relationships could exist between these parameters due to the interplay between the various aspects of the application execution including analysis tasks, data and the systems. The next goal of the thesis is to augment existing specification capabilities
for applications with high-level descriptions (or annotations) of performance parameters. Metadata describing multi-dimensional datasets and the functional properties of analysis tasks must be expressed in a system-comprehensible form for their use in application configuration decisions. To allow users to specify such information at a high level, this thesis proposes the use of a conceptual knowledge model that can capture the identified parameters, and the relationships between the parameters and the performance (and accuracy) of analysis. To implement this proposed model, I borrowed techniques from the semantic web world, including ontologies and rules to represent the necessary concepts and relations. The abstract workflow description is now augmented with these additional metadata descriptions and domain-specific rules to address the configurability requirements of this class of applications [63].

Prototype implementation for application configuration and execution. Finally, we present IMPACT, an integrative framework to enable configuration of multi-dimensional data analysis applications for efficient execution from a performance point of view [64, 70]. The earlier contributions of this thesis are helping to lay the groundwork for this framework. The framework is a proof of concept for the proposed idea of having the system translate high-level conceptual knowledge about application performance parameters into low-level execution plans for different middleware layers in order to meet the application challenges presented in section 1.2. A particular emphasis of this framework is on supporting various accuracy-performance trade-offs in multi-dimensional data analysis. Figure 7.1 shows the architecture of
the IMPACT framework which has been developed as a collaborative effort involving research groups from the Information Sciences Institute at the University of Southern California and the Compiler Research Group at the University of Utah. We also present a prototype implementation of the framework which allows the expression and user-driven tuning of multiple performance parameters for two real-world applications.

Figure 1.7: Design of framework to support performance optimizations for large scientific multi-dimensional data analysis

1.4 Organization of the Thesis

The remainder of this thesis document is organized as follows: Chapter 2 presents the motivating applications for this work and makes a case for conceptual model of such applications borrowed from various domains to gain a better understanding of their performance requirements. Chapter 3 describes system support for large-scale scientific data analysis and presents my approaches to providing a layer of multi-dimensional data services to applications. In chapter 4, I describe application-level configuration services that can be developed to resolve mismatches between analysis requirements and heterogeneous systems to meet time and resource constraints. Chapter 5 introduces the notion of accuracy-performance trade-offs in data-intensive scientific applications and demonstrates the potential impact of such trade-off services when employed to meet different kinds of application-level quality-of-service (QoS) requirements imposed by the user. Chapter 6 consolidates the learnings from the previous three chapters to compile a summary list of configurability requirements for data-intensive applications and discusses language support to enable the high-level specification of these requirements to the system. Chapter 7 presents a proof-of-concept prototype implementation of my proposed framework called IMPACT and describes my experiences through experimental evaluation of real application use-case scenarios. Finally, this document concludes with a list of lessons learned in this thesis, and provides insights into how the work could be extended to develop systems which can incorporate domain-specific performance optimizations.
CHAPTER 2

MOTIVATING APPLICATIONS AND SCIENCE REQUIREMENTS

The challenges presented in Chapter 1 briefly alluded to some of the motivating scientific applications that are likely to benefit from the contributions of this thesis. In this chapter, I explain this class of applications and the defining characteristics of the application workloads in greater detail, and also describe the different kinds of science requirements that arise within these applications. The increasingly data-driven nature of scientific research means that analysis of scientific data is akin to finding a needle (the useful information) in a haystack (the massive datasets). Before attempting to provide solutions for efficient analysis, it is important to conceptually model these applications to better understand the different factors that affect their execution.

2.1 Application Motifs: A case for conceptual models

Prior attempts to classify applications into well-defined categories or motifs [16, 17] are based on patterns in computation and data movement that allow reasoning about their behavior under different execution conditions. Specific algorithmic
instances within these application motifs may change over time, but the underlying patterns that characterize them remain the same. Figure 2.1 shows twelve motifs identified based on the computational patterns of some common applications. The defining characteristics of certain motifs such as dense/sparse linear algebra and spectral methods are compelling enough that dedicated numerical libraries such as ScaLAPACK [35], ATLAS [109, 110] and FFTW [44] have been developed to address their specific execution requirements. These libraries can optimize the common case (e.g., important kernels) on specific system architectures. That is, the low-level computation and data communication operations for applications within this motif can be customized for optimal performance depending on the system. This is possible

Figure 2.1: Grid showing twelve application motifs identified from computational patterns within different application domains. Color of a cell represents the relative presence of the pattern within the domain – red(high), blue(low). Courtesy of Berkeley Par Lab
because the libraries are developed based on conceptual models of numerical analysis applications that accurately capture their characteristics.

Not many attempts have been made to categorize data-intensive applications within different domains based on their computation and data access patterns. In recent years, the systems community has converged upon the MapReduce [38] model, developed at Google, as an important application primitive that can be used to conceptually describe the requirements of a large set of applications that analyze Internet-scale data. The primary computational characteristic of this model is data-parallelism, i.e., a single ‘map’ function executes in parallel on independent collections of input data to generate intermediate key/value pairs, which are then eventually combined using a single ‘reduce’ function that also operates in parallel on the intermediate data to reduce them to a single or smaller number of results.

Despite the simplicity and wide applicability of such a conceptual data-parallel model, it does not sufficiently represent all irregular computations and complex data access patterns within applications. In fact, the MapReduce model actually is just one instance of the broader application motif of ‘generalized reduction’ operations that manifest themselves in several application domains. The Active Data Repository project [71, 72] proposed a model and a runtime system to support generalized reduction operations that mapped data from one (input) multi-dimensional data space to another (output) space. The FREERIDE [47, 56] project provides language and runtime support for generalized reduction style operations within the domain of large-scale data mining applications. Attempts to classify the other kinds of applications based solely on the data movement patterns have suffered from a lack of generalizability of the application structure. My research seeks to identify additional motifs or
sets of computational patterns and data management requirements of data-intensive applications. In particular, my research seeks to put forth a conceptual model that can describe the domain of high-resolution scientific multi-dimensional data analysis applications at a high level of abstraction based on knowledge gained from the requirements of applications within this domain.

The remainder of this chapter presents several application areas that require the analysis of large scientific multi-dimensional datasets, and well-defined application instances borrowed from these domains. I will then attempt to characterize these applications conceptually and identify new motifs to contribute towards the specification goal of this thesis.

2.2 Target Application Domains

In this section, I discuss motivating applications from different domains that are expected to benefit from my contributions.

2.2.1 Biomedical Imaging

Biomedical imaging plays a crucial role in basic research and clinical studies in biomedicine. In recent years, researchers from different fields of study in pathology have enjoyed rapid advances in such imaging modalities as functional Magnetic Resonance Imaging (fMRI), digital high-power light microscopy, and digital confocal microscopy which allow detailed measurements about biological entities to be collected rapidly. Information synthesized from such measurements is used in the diagnosis of deadly diseases, assessment of the effectiveness of treatment regimens, and the study of complex biological processes. The use of digitized, whole-slide pathology imaging systems reduces the dependency on manual observation of glass slides, and replaces
it with a virtual microscopy environment that provides reliable and reproducible results. High-resolution images obtained from pathology slides and tissue microarrays generate a wealth of information about the microscopic characteristics of organisms. The in-silico analysis of such images allows the study of normal and disease states of biological systems at cellular and subcellular scales.

**Hardware:** Advanced microscopy instruments capable of generating high-resolution images are commercially available. The Aperio ScanScope XT digital slide scanner can automatically scan 120 slides at a time at up to 40x magnification [2]. The DMetrix DX-40 scanner can produce ultra-high-resolution scans of whole slides at 20x magnification in under a minute [5]. The hardware in these instruments continues to develop at very rapid rates. In my opinion, the mass production of such scanners will make them more affordable over the years, and their commercial viability will make them a necessary part of research in biomedical imaging. An average observation in 3D confocal microscopy may involve the scanning of digitized image slides of a tissue sample, at up to 150x magnification. At high resolutions, it may not be possible to generate an image of the entire sample at once, because of the limited field of view (FoV) of the instrument. In such cases, advanced scanners capture the entire image as a set of smaller subimages or tiles, where each tile corresponds to a rectangular portion of the image scanned at that resolution. Some scanners like DMetrix adopt array-based scanning with multiple sensors mounted on the scanner head to simultaneously capture high-resolution images of different portions within an FoV. The instrument stage may also move the imaging sensors in X and Y directions in fixed steps. At each step, multiple snapshots of the rectangular subregion (e.g., a $256 \times 256$ square tile of pixels) under the sensor head are captured at different
focal lengths (different points in the Z dimension) to obtain a 3D snapshot of tissue content.

**Data and Analysis:** Typically, a single scan produced by a state-of-the-art scanner today may be tens of Gigabytes in size ($\gtrsim 150k \times 150k$ pixels). Data sizes increase when we factor in the multiple focal planes. Aperio has already demonstrated the capability of scanning Terapixel images (of breast tissue) using their ScanScope scanner – each pixel could be represented by a 3 byte vector for its RGB color values. Today’s high throughput scanners can produce raw image data at the rate of about 3 to 5 Gigabytes per minute. In addition to image pixels, the raw data may also contain metadata describing the acquired datasets. Comparative analyses of tissue samples obtained from a large cohort of subjects could result in multiple hundreds of Terabytes of image data. Image analysis operations on these large volumes of data could range from a simple 2D visualization or browsing of the images to more complicated operations (like 3D reconstruction of the specimen under study from multiple 2D slides) all arranged in the form of an analysis workflow. The following use-case scenarios borrowed from real applications provide a good idea of the kinds of analysis requirements in biomedical imaging.

**Use-case scenario 1: Pixel Intensity Quantification (PIQ).** The distribution of signal (color intensity) values of pixels across different regions of interest within an image can be used as biomarkers to gain insights about disease mechanisms (Figure 2.3(a)). For example, a region with extremely high intensity red pixel values may indicate the presence of cancerous cells, while a region characterized by change from high red to green pixel values over time may indicate that the prescribed treatment
is working successfully for the patient under study. This requirement translates to quantifying the aggregate pixel intensity within the different polygonal regions of interest specified by the researcher. Figure 2.2 displays the formulation of the target aggregation queries for one such region as a relational database query. Given a

\[
\text{SELECT } \text{Aggregate}(Value(x,y)) \text{ FROM Image}
\]

\[
\text{WHERE } (x,y) \text{ in POLYGON} < (x_1, y_1), \ldots, (x_n, y_n) >
\]

Figure 2.2: Formulation of aggregation query as a relational database query.

region of interest, this query request aggregates the results of Value of each point in that region, for a predefined Aggregate function. To obtain the desired pixel intensity values within the regions of interest, neuroscientists at the National Center for Microscopy and Imaging Research at San Diego designed a workflow for processing images of mouse brain tissue samples scanned using digital confocal microscopes (Figure 2.3(b)). This analysis workflow quantifies pixel intensity within user-specified regions through a series of raw data correction steps as well as thresholding, tesselation, and prefix sum generation operations. That is, the above class of queries can be executed more efficiently if the image data is preprocessed. More specifically, if the aggregation functions are Sum or Count, which are the usual target aggregation operations, one can generate 2-dimensional (2D) prefix sums to answer queries efficiently [107].

A detailed description of each step in this workflow can be obtained from [36, 107] and is beyond the scope of this document. The zproject phase aggregates multiple Z planes of a multi focal plane input image into a single output image plane, using
the $\max$ aggregate operator. The \texttt{prenormalize} phase computes average and offset intensity tiles for the projected plane. \texttt{Normalize} corrects for the variances in illumination that may exist between tiles and is critical to creating a seamless mosaic of tiles. The \texttt{autoalign} phase determines how the distinct partially overlapping tiles should converge upon one another to produce a properly aligned image, eliminating all overlapping regions. The output of this phase is a list of local displacements for every image tile. Reinterpreting each tile as a graph node and each displacement as a graph edge, the \texttt{mst} phase computes the maximum spanning tree of the graph (maximizing the displacement score) to determine the best global alignment across all tiles and to produce finalized offsets for each tile in the output image. The \texttt{stitch} phase combines the partially overlapping tiles based on the finalized offsets, producing a final non-overlapping (stitched) image mosaic. The \texttt{warp} step geometrically transforms an image to fit a standard appearance or a pre-defined canonical atlas (e.g., the Allen Reference Atlas (ARA) – a high-resolution anatomical 3D atlas of the mouse brain [1]), based on a set of control points provided by the user. There
are three preprocessing tasks in the pipeline: The **thresholding** step reads the input image data and simply overwrites each color value (Red, Green, Blue) with a 0 or 1, depending on whether the value is outside or inside a user-defined threshold. The optional **tessellation** step reads thresholded pixel data as input and produces tessellation output, which consists of the sum of all nonzero pixels within a user-defined tessellation region. The tessellation region is specified by the user in terms of an $x$ and $y$ range, and $x \times y$ becomes the factor by which the image resolution is reduced. The **prefix sum generation** step computes and stores the 2D prefix sum of the values at each point of the tessellated image. The 2D prefix sum at each point $(x,y)$ of the tessellated image is given by:

$$PSum(x, y) = \sum_{i=1}^{x} \sum_{j=1}^{y} value(i, j)$$

These prefix sums are subsequently used for efficient execution of aggregation query operations [107].

**Use-case scenario 2: Characterizing Phenotype Change (CPC)** The quantification and visualization of structural phenotypes in tissue samples helps understand how genetic differences can affect the structure and function of multi-cellular organisms. For example, the deletion of the gene retinoblastoma (Rb) is believed to cause extensive morphological changes in mouse placenta including possible reduction of total volume and vasculature of the placental labyrinth, increased infiltration from the spongiosotrophoblast layer to the labyrinth layer, and clustering of labyrinthic trophoblasts [80]. To quantitatively ascertain these effects, the 3D structural attributes of microscopic tissue specimens must be constructed from a large dataset consisting of 2D histology image slices and then evaluated and compared against a non-diseased
mouse placenta tissue. Researchers at the Ohio State University put forth an analysis workflow (as shown in figure 2.4) to enable the study and characterization of phenotypical changes in the tissue. The same high-level analysis stages could be applicable regardless of the specific tissue and gene under study.

2.2.2 Astronomy Sky Surveys

Sky surveys have led to some of the most important discoveries in astronomy \[6 \, 10, 12, 11\]. For several centuries, numerous sky surveys were undertaken by astronomers to map (i.e., generate atlases of) the night sky by identifying various celestial objects. The technology to conduct these surveys improved dramatically in the 20th century. Modern sky surveys are large-scale initiatives (each with their own observatories and specialized telescope instruments) that do more than mapping of visible objects in the sky; they are extensive, wide-field efforts intended to map distant galaxies and the entire universe. Astrophysicists use the data gathered by these instruments to look for dangerous asteroids, black holes lurking in the centers of galaxies, exploding supernovae, mountain masses within stars, quasars and brown dwarfs – the most distance objects in the universe. Images from the telescope instruments are processed to yield photometrically and astrometrically well-calibrated data that enable the identification and classification of celestial phenomena.

**Hardware:** The Two-Micron All-Sky Survey (2MASS)\[5\] and the Sloan Digital Sky Survey (SDSS)\[6\] are two prominent sky surveys from the last decade that generated enough data to foster the development of specialized software for data management and analysis. 2MASS used two highly-automated photometric telescopes,

\[5\]http://www.ipac.caltech.edu/2mass
\[6\]http://www.sdss.org
Figure 2.4: Data analysis workflow for studying Phenotypical change
each equipped with a three-channel camera, each channel consisting of an array of plate detectors, capable of observing the sky simultaneously at three different infrared wavebands. With the advent of arrays of CCDs in cameras, surveys like the SDSS are able to not only map “faint” objects in the universe at five different wavelengths, but also gather measurements about their positions and brightness and the relative distances between objects very precisely. The SDSS telescope captured images of stripes of the night sky every night, and each stripe was imaged at five different wavelengths. The Large Synoptic Survey Telescope (LSST) is a wide-field reflecting telescope that will photograph the entire sky every three nights. The 8.4m LSST has the world’s largest digital camera, and can cover large regions of the southern sky at high resolutions in a single exposure.

Data and Analysis: Every new wide-field sky survey initiative seeks to push the envelope with respect to the how clearly scientists can see objects at the deep end of the universe. As a result, these surveys are characterized by unprecedented data acquisition rates. For instance, the LSST has a 3.2 Gigapixel camera that is programmed to capture an image of a sky region every 15 seconds. Given this rapid cadence of the observing system, this translates to 20 Terabytes of raw image data acquired every night. The archived image data is expected to amount to 60 petabytes after 10 years. Each image corresponds to a field-of-view of the telescope (~ a 10 sq. degree region of the sky) and is around 6.4 Gigabytes in size. Objects extracted from the images are stored in catalogs. These periodically updated catalogs are expected to contain around 50 billion objects at the end of the survey. The LSST uses this acquired data to meet several science goals. Transient alert generation is one such
goal that adequately represents the analysis requirements of large sky surveys.

Use-case Scenario 3: Transient Alert Generation (LSST) One of the primary goals of the LSST survey is the near-time generation of alerts when unknown transient objects or events are spotted in the sky. That is, for any new object that has not been previously observed, a decision must be made to trigger an “alert” so that astronomers can be notified immediately thereby leading to closer monitoring of the transient object [21]. The alert generation will involve reduction of the raw image data obtained from the telescope. Astrophysicists from the LSST have put forward a data analysis workflow, shown in Figure 2.5 to generate alerts and other data products from the raw images.

![Figure 2.5: Data analysis workflow for use-case scenario 3 (LSST)](image)
The inputs to the pipeline are multi-Gigabyte sized images that arrive from the telescope every 60 seconds. These images first undergo a series of low-level image processing steps for photometric calibration and to account for other defects introduced during their acquisition. Determining the World Coordinate System (WCS) for the image is part of astrometric calibration. The most important among the image processing tasks is the determination of an appropriate Point Spread Function (PSF) model. This may require refinement of the WCS in an iterative manner until an acceptable PSF is obtained. Then, the difference imaging kernel is built using both the astrometric and photometric registration of the image. Finally, this stage also generates quality metrics for the image such as PSF quality, camera performance, atmospheric transparency etc., and feeds them back to the observatory control system. The detection tasks are responsible for detecting light sources in the image using differential image analysis and store these detections in a catalog. One may have to remove entries in the catalog corresponding to cosmic rays or satellites. The association tasks then compare these detections to historical information about objects in the sky, and try to classify them based on their similarities to the objects. The mapping of detections to objects based on their positions in space is carried out by the spatial crossmatch step. The final classification is made based on other parameters and includes probabilistic association techniques. Alerts are raised any time a detection cannot be classified as one of the known objects. It is expected that LSST will generate some 100 thousand alerts per night [22].
2.2.3 Particle Physics

Particle physics is the study of the basic elements of matter and the forces acting among them under different conditions. Experiments and simulations within this domain model the effects of collisions and interactions between atomic and sub-atomic particles at very high energies or under other extreme conditions and produce massive amounts of measurement data [20]. A set of post-processing analysis and visualization operations are then performed on the simulation output to study the dynamics of the particles and determine important structural and functional properties of the entity under investigation. For example, researchers investigating the feasibility of a new material to construct spaceships may wish to study the effects of extreme temperature and pressure conditions on the dynamics of its constituent molecules. Other scientists may wish to investigate the theoretical foundations behind turbulence transport or combustion propagation under different experimental conditions or the conditions that prevailed just after the Big Bang occurred billions of years ago. Such studies are only possible through large-scale simulations or experiments conducted using advanced accelerators.

Hardware: The accelerator is the basic tool of particle physics. It can orchestrate particle collisions at high energies. Accelerators work by accelerating charged particles using electric fields. A linear accelerator accelerates particles in a straight line. The Stanford Linear Accelerator at SLAC is two miles long

7\[http://www6.slac.stanford.edu/Research_AcceleratorPhysics.aspx\] Circular machines are more common. As well as accelerating the particles using an electric field, circular accelerators bend their paths using a magnetic field as in the Large Hadron Collider
When they have opposite charges, the particles being accelerated travel in opposite directions until they are forced to collide. The LHC lies in a tunnel 27 kilometres (17 mi) in circumference, as much as 175 metres beneath the surface. Where the physics cannot be experimentally realized, scientists mainly rely on petascale simulations conducted on high-end supercomputing systems. For example, in combustion studies that seek to understand how a lifted autoignitive flame is stabilized in a canonical configuration, the direct numerical simulation of the near field of a 3D turbulent lifted jet flame at 1550K is performed at a jet Reynolds number of 10,000 with 1.28 billion grid points over 18 global time steps. Many high-end supercomputing systems such as the Cray XT-series machines [96] and Anton [98, 99] are created specifically for the purposes of carrying out such simulations in different domains.

Use-case scenario 4: Gyrokinetic Toroidal Code (GTC). The Gyrokinetic Toroidal Code (GTC) is a 3D Particle-In-Cell simulation code used to study micro-turbulence in magnetic confinement fusion from first principles plasma theory [42]. The data space used to model the particle data includes two 2D arrays (for electrons and ions respectively). Each row of the 2D array records eight attributes of a particle including coordinates, velocities, weight, and particle label. As the simulation evolves, the particles undergo perturbation and are displaced from their original positions in the space resulting in an out-of-order particle array. The GTC simulation can output two million particles roughly every 120 seconds resulting in 260GB of particle data per output. This output is in the form of HDF5 files. Complex visualization tasks

such as isosurface rendering may be performed on the simulation output. As part of
the post-processing, several preparatory analysis tasks may be performed on particle
data output ahead of the visual exploration. Examples of such tasks include tracking
particles of a million-particle subset out of the billions of particles across multiple
iterations, range queries seeking particles that lie within specific ranges, and the
construction of 1D and 2D histograms on the particle attributes to enable interactive
online monitoring and visualization. 2D histograms are also used for visualizing
parallel coordinates in subsequent analysis tasks[116].

2.2.4 Satellite Imagery

Satellites orbiting the earth are fitted with several sensors that can continuously
capture images of the earth’s surface in multiple wavelengths of electro-magnetic
radiation (multi-spectral). Multi-spectral platforms such as the Landsat thematic
mapper then process these images in order to reveal geographical and environmental
information related to monitoring of land usage, deforestation, prospects for mineral
discovery in a region and so on. Other themes concern the nature of meteorological

---

Figure 2.6: Data analysis workflow to process Satellite Image data from multiple
sources
phenomena including hurricanes, floods etc., time-critical events that require real-time analysis of large volumes of surveillance data gathered from the satellite sensors. Interest in satellite imagery was further spurred by freely available software such as Google Earth that allow exploration and visualization of high-resolution 2D and 3D images of any part on the earth’s surface. Along similar lines, there also exist software that helps explore the surfaces of other planets like Mars, and also the surface and terrain of the earth’s ocean bed at high resolutions. TerraServer-USA\(^9\) is a web-service that provides free public access to the world’s largest online database, containing terabytes worth of data, of maps and aerial photographs of the United States. Several satellite imagery and remote sensing efforts are hitting the terabyte barrier these days. Figure [2.6](#) is an example of an analysis workflow borrowed from this domain.

2.3 Characterizing Application Workloads

The distinguishing characteristics of data-intensive scientific applications common across multiple domains can be summarized as follows:

1. **High-resolution multi-dimensional datasets.** In our target application domains, the principle source of data are either advanced digital sensors and scanning instruments or petascale or larger simulations. The raw datasets generated by each source basically consists of a large number of data points that correspond to measurements or recordings of attributes over some large multi-dimensional space used to model the phenomena under observation. A subset of these attributes correspond to the spatial coordinates of the points that make

\(^9\)http://www.terraserver-usa.com
up the space – Raw image data slices acquired from scanning instruments are an instance of a 2D space of pixel points, although some imaging modalities such as digital confocal microscopy produce 3D image data volumes that span several focal planes. Individual data points (be it pixel data or particulate data such as atoms and ions) are typically associated with a low number ($\approx 10$ to $20$) of non-spatial measurement attributes that make up the remaining dimensions. These attributes could include scalars representing, say, the momentum and velocity measured at that point, or vectors, say, representing RGB color intensity values for that pixel. An important distinguishing characteristic of these datasets is their high spatial and temporal resolutions. Raw image data could contain trillions of pixel points, while 3D atomistic simulation models are based on hundreds of billions of interacting particles in space. Data collections are acquired rapidly over time, and this temporal resolution adds another important dimension to these datasets. In astronomy and satellite imagery, high-resolution photographs of entire sky or earth regions are captured repeatedly every two to three nights. This high temporal resolution allows the study of variations in the phenomena over time.

2. **Legacy file formats and generic data models.** Given the multi-dimensional nature of large scientific datasets, the lack of conformance with the traditional relational models in database systems meant that researchers in different domains came up with customized file formats for the storage of the raw datasets. Examples of such well-known formats include netCDF [74] and HDF5 [48] for storing experimental data from climate modeling simulation outputs, the FITS
file format for astronomy data, ROOT for high-energy physics data, and SVS for storing digital pathology image data. Prior to analysis in a parallel computing environment, it is important to understand the data models used to represent the data entities within such datasets. Raw data from the flat files are usually transformed into an object-relational model for analysis purposes and to provide easy access to individual data points. The models used for representation of data within a multi-dimensional space could vary depending on the domain. In experimental datasets obtained from simulations, the data points within each iteration are assumed to be arranged in a Grid or a lattice, and are represented using meshes. These meshes in turn could be uniform tetrahedral or hexahedral, and distances and interactions between points are measured based on the Euclidean coordinate system. In observational datasets, image data can be considered to be a dense multi-dimensional array of pixels or voxels and are similar to meshes in this regard. On the other hand, in astronomy surveys, the sky is typically modeled as a sphere where celestial objects are associated with positions within the equatorial coordinate system. If the underlying data representation remains fairly constant over time and is well-known a priori (e.g., the spatial extents of the observed sky will not change over time), then such metadata information could potentially be exploited to enhance the execution of analysis operations on the data.

\( ^{10} \text{http://fits.gsfc.nasa.gov/fits_intro.html} \)

\( ^{11} \text{http://root.cern.ch/root/InputOutput.html} \)
3. **Workflow-based analysis.** Most large-scale scientific applications are expressed as interdependent data analysis pipelines consisting of tasks with well-defined data flow dependencies between them (collectively referred to as a workflow). These tasks or components of a workflow are usually developed independently, and could be written in multiple languages. In most cases, these components are not developed from scratch – instead, the application workflows are composed from large libraries of reusable components and existing workflows within a domain. Workflows are designed to manage the data end-to-end in that any processing operation performed on the data is identified with its own specific task within the workflow; these may include data acquisition, transformation into a multi-dimensional data model for analysis, different forms of analysis tasks, data archiving and visualization. Based on observations from target application domains presented in this chapter, Figure 2.7 presents my abstract view of a typical workflow for multi-dimensional data analysis. Here, the raw input datasets from the source(s) are first transformed through a series of operations that extract the data from files and transform them into a form suitable for analysis tasks that follow. In the case of sensor (observational) data, these transformation operations are also responsible for eliminating errors introduced into the data by incorrect calibration and movements of the instrument during data acquisition. The analysis tasks in the workflow may search the transformed data in order to find interesting regions with respect to user-specified criteria, or find similar regions across multiple datasets, or selectively identify favorable or abnormal regions within the data, or any other operation
Figure 2.7: Generic workflow template for scientific multi-dimensional data analysis
that processes the data space as per user requirements. A single workflow instance may operate on input datasets coming from multiple sources. Multiple workflow instances within an application execution can share the same input datasets.

4. **Generic user-defined analysis operations.** Analysis workflows across multiple application domains are too variable for them to be generalized. For example, a workflow to identify galaxies in the sky may not be directly used to identify hot-spot regions in particle simulations even though the 3D input data in both cases share similar characteristics. This lack of generalizability suggests that workflows designed for use in one domain address some key analysis goals and hence, may not be assumed to be directly applicable in other domains. However, commonalities can be drawn at the level of individual operations or primitives that make up these workflows. Operations performed on scientific multi-dimensional datasets are typically defined by the user in some high-level language. These operations could be categorized into *low-level data processing* tasks and *high-level data analysis* tasks. Low-level processing tasks operate on raw data to reduce them to one or more intermediate data forms as required by the downstream analysis tasks; in turn, the high-level tasks operate on the intermediate data to identify domain-specific data objects, features, and other interesting regions within the datasets, the annotation and study of which may lead the scientists to the desired insight and knowledge.

Low-level processing tasks constitute the initial part of most application workflows. Raw multi-dimensional datasets are transformed into intermediate data products through these tasks. The common aspect to these tasks is that
they typically view input datasets as a collection of data points in some n-dimensional data space. Each data point is a zero-dimensional spatial entity (e.g., a pixel or an ion) in that it has no spatial extent and is just a point in space that is associated with a vector of spatial coordinates. Each such task can be represented using an arbitrary function $f$ mapping data spaces as follows:

$$\text{Output}([OD_1, OD_2, \ldots, OD_m], [OX_1, OX_2, \ldots, OX_n])$$

$$\rightarrow f(\text{Input}\{<ID_1, ID_2, \ldots, ID_p>, [IX_1, IX_2, \ldots, IX_q]\}) \quad (2.2)$$

That is, each low-level task takes as input multi-dimensional data spaces $ID_1, \ldots, ID_p$ where $p \geq 1$, and an optional set of auxiliary inputs $IX_1, \ldots, IX_q$ and applies some transformation function $f$ on the input data. This is a user-defined (and sometimes domain-specific) operation that produces as output a set of multi-dimensional data spaces $OD_1, \ldots, OD_m$ and auxiliary outputs $OX_1, \ldots, OX_n$ where $m, n \geq 0$.

High-level analysis tasks are responsible for accurately modeling human perception of data entities present within transformed ‘intermediate’ data in the workflows. These tasks view data as a collection of domain-specific entities including data objects and features in some n-dimensional space. Each data object corresponds to some human-perceivable $k$-dimensional spatial entity ($k > 1$) within the domain and has some spatial extent (e.g., line, polygon, sphere). For example, in biomedical imaging, cells and nuclei could be characterized by circular shapes, while lesions could have more of an elliptical shape. These objects are

---

12Auxiliary input is used to refer to any additional inputs and metadata that are not directly generated by the data sources, but are instead furnished by the users. These are typically not multi-dimensional in nature and are orders of magnitude smaller than the data spaces.
associated hundreds of dimensions or features such as radius and eccentricity that help characterize their shapes comprehensively. Processing operations performed by these tasks include data clustering and mining, and computations for feature identification, classification and tracking of objects. In the case of simulation data, the visualization of the data objects and their features is important for the scientist. Hence, within such domains, it is common to find workflows where visualization tasks and quantification analysis tasks are interleaved.

5. **Adaptivity.** An important characteristic of operations that process spatial, multidimensional data is that of *adaptivity*. That is, in cases where the users are tolerant to errors in the output, adaptive operations can operate at different performance levels; i.e., they can selectively process input data entities in a manner that trades accuracy of analysis for improved performance. Chapter 5 elaborates on this characteristic of the application workloads.

6. **Science requirements.** Application workflows from different target domains generally have different science requirements. The basic requirement of all applications is to minimize the end-to-end execution time, i.e., the time required to execute all tasks in the workflow to produce the desired end-result. Real-time and near-time constraints are present in certain application domains. For example, in time-critical applications (of which the LSST use-case scenario is an example), certain events, such as the detection of an abnormality in the dataset, must trigger alerts in real-time to direct the attention of the scientist to specific portions of the dataset where the abnormality occurred. Real-time feedback from the analysis workflow will help instruments track the movement
of these potentially transient data objects within the multi-dimensional space. The next set of observed requirements within the applications concerns the nature of execution of the tasks – for certain tasks, especially visualization, users may require an interactive mode of execution so that they can steer the future analysis tasks based on intermediate results they see. This requirement for online monitoring of intermediate results is in contrast to the batch mode of execution of applications where users submit input datasets and obtain results only when all analysis tasks have completed their execution. Additional requirements from the users may concern the quality of the results produced by the analysis tasks. For some analysis goals (such as feature-based classification), a perfect result may not exist. In such cases, the users are interested in obtaining the best possible result within the constraints of time. Typically, each analysis task is bound to a piece of code during execution. However, in certain scenarios, especially with high-level analysis tasks, the user may require that the task be bound to multiple codes (e.g., evaluating multiple models to assist in the classification), and that the result for the task be obtained by combining the results from each code in pre-specified ways.

2.4 Summary

The early part of the chapter discussed how conceptual modeling of numerical methods within various application domains based on their computation and data movement patterns have led to their classification into well-defined motifs. High-level models enable the development of custom performance optimizations in software for
specific motifs. Data-intensive scientific applications process high-resolution multi-
dimensional datasets through workflows of user-defined analysis operations. These
operations could be deployed in serial or parallel. Workflows could be sequential
pipelines or DAGs or iterative in nature. The classification of these applications
based solely on analysis patterns is too large in scope to be useful given the arbitrary
structure of workflows and the high variability among workflow operations across dif-
ferent domains. Previous attempts to identify well-defined workflow analysis patterns
as motifs have not led to the development of specialized software suites (a notable
exception being MapReduce for generalized reductions).

In my opinion, based on observations of scientific multi-dimensional data analysis
applications, seemingly different operations from multiple domains can be grouped
together based on how they map data spaces, or the kinds of operations they perform
on the data. For example, the PIQ workflow in use-case scenario 1 requires com-
putation of a 2D prefix sum over image data, while the GTC workflow in use-case
scenario 4 requires the computation of 2D histograms over particle array data. Both
of these operations functionally represent the computation of global data structures
across entire datasets to assist in subsequent analysis, and may thus have commonal-
ities in their requirements. As another example, image warping in use-case scenario 1
involves mapping each point in one data space (the image) to a point in another space
(the atlas). Similarly, in particle studies, displaced atoms in one frame may need to
be mapped to their respective positions in another frame from a previous point in
time. Both these operations map data spaces on a per-point basis, and therefore may
share similar requirements. Therefore, instead of seeking to classify entire applica-
tion workflows based on their structural patterns, I believe a conceptual model for
scientific multi-dimensional data analysis applications must include high-level specifications for different aspects of the workflow (tasks, links, ensembles or sub-workflows) classified based on their functional requirements. For example, tasks where multiple computational models may need to be simultaneously applied to study a given input may belong to one motif, while ensembles where data needs to be streamed from one task to another may belong to another motif. It is often the case that the high-level functional requirement(s) of a motif would need to be translated to low-level execution configurations depending on the underlying system. A conceptual model for specifying characteristics of applications motifs will help in the automation of this translation process.
CHAPTER 3

SYSTEM SUPPORT FOR LARGE-SCALE SCIENTIFIC DATA

Developments in the commodity components that constitute today's systems allow applications to deal with large data – cheaper disks make it possible to assemble storage array subsystems consisting of multiple disk drives that can house terabytes of data per node; network bandwidth within HPC infrastructures has dramatically improved over the years with the advent of 10-Gigabit Ethernet and switched fabric communications like Infiniband that provide highly scalable connections between compute nodes and storage devices. On the other hand, the amount of physical memory per node and the disk bandwidth to local disks on each node within these systems have not seen commensurate developments, and these limit a system's ability to scale gracefully in the face of increasing data. Software support is required to bridge the gap between application demands and the systems brought about by these limitations. This chapter documents my efforts at providing the necessary software support to address the first challenge of data-driven scientific applications, namely, the large data scales. Specifically, I describe my experiences in providing I/O virtualization for these applications in a shared-nothing environment via a layer of parameterizable multi-dimensional data services, and related performance optimizations.
3.1 Background

Software support to manage large collections of data can be classified into different categories. In this section, we briefly review these categories and their existing level of support for large-scale scientific, multi-dimensional datasets.

**Distributed File Systems** allow access to disk-resident files shared among multiple nodes via a network. These file systems implement the necessary protocols for indexing and accessing distributed data in a transparent manner. Applications do not have direct access to the underlying data block- or object- storage on disks, but instead, read from and write data files onto distributed storage. The primary focus of these systems is to provide massive scalability (upwards of several petabytes of data) and high availability of data at such scales through various fault tolerance mechanisms and transparent replication, rather than low latency of data access. Traditionally, distributed file systems relied on slow local area networks and suffered from poor performance. Newer object-based distributed file systems deployed on large clusters are specifically optimized for high performance storage and access to data. Most parallel, distributed file systems stripe data over multiple servers and provide for concurrent access by multiple processes of a parallel application in an effort to improve performance [27, 4, 108]. Lustre [4] is one such highly scalable file system which can support up to tens of thousands of client systems, petabytes of storage and hundreds of gigabytes per second of I/O throughput. Google File System (GFS) [45] is a proprietary distributed file system developed by Google to optimize storage and access to enormous amounts of data on large clusters. Hadoop’s underlying file system HDFS [25], the open-source version of GFS is a distributed file system that also stores data as large files (several megabytes) and transparently replicates data across multiple hosts.
A majority of scientific data is stored in the form of files. Hence, distributed file systems may appear to be a natural choice to handle their storage and access on large clusters. However, the multi-dimensional nature of scientific data and the data access requirements within different application instances present performance challenges that most distributed file systems are not designed to address out-of-the-box. Despite optimizations for file organization and retrieval in these systems, they may not readily provide the desired levels of flexibility to support optimal execution of scientific applications. Moreover, the consistency protocols and fault-tolerant metadata (and data) management at the level of individual objects may add undesirable overheads to the application execution when large amounts of data are accessed frequently. Several specialized I/O subsystems have been developed on top of existing distributed file systems to support efficient I/O for scientific multi-dimensional data in large shared-nothing cluster environments. Such systems often include their own on-disk format for storing data, run-time libraries and tools for optimizing access to data by specific application instances. Parallel HDF and parallel netCDF [74] are examples of runtime support libraries that can efficiently handle large scientific datasets stored in HDF5 and netCDF formats respectively. However, the fact that each library is tied to a specific data format and the optimizations are tightly coupled to a specific architecture prevent its applicability to different application domains. To overcome this dependency, ADIOS [78] provides a componentization of the I/O subsystem, where a single API allows applications to customize different libraries and data I/O routines depending on the dataset characteristics and requirements. Like HDF5, ADIOS provides a data format called BP, that allows the association of rich application-level metadata with each block of binary-packed data. The important
point here is that the I/O subsystems use application-level metadata in order to tune the underlying file systems for efficiently managing scientific data.

**Federated, Global-scale File Systems** are closely related to distributed file systems, with the exception being that storage and access is managed across disparate clusters and a large number of servers and workstations over the wide area. That is, the federation is web-scale or Grid-scale. These storage resources may be located within the same data center, or may be located at geographically disparate locations. In Cloud-based infrastructures, storage systems such as Amazon’s S3 [89] seek to provide the levels of scalability and support high-speed file system-like operations and hence require such global-scale file system support. Since no single central site can be assumed to maintain metadata at a global scale, the underlying architectures of these file systems is similar to a structured peer-to-peer system of nodes. Peer-to-peer systems support storage and secure sharing of content and files across disparate data sources and applications that form a peer-to-peer network. These systems provide support for location independent routing and asynchronous data replication in a distributed environment. Architectural and middleware support for data management (including data replication and protocols for high-speed data transfer) in such systems have been addressed in several efforts [33, 43, 104]. Database-like storage and retrieval for scientific datasets in Grid systems has been supported in a manner that is independent of legacy data formats [84, 77, 31].

**Parallel Database Systems** [40, 18] running on multiple nodes were the subject
of extensive research over the decades and laid many of the foundations for exploiting parallelism in data analysis including data partitioning, intra-query pipelined and partitioned parallelism between data operators and hash-based data distribution among processors. Several research prototypes of parallel DBMSs such as Gamma and Bubba architectures [41] from the 1980s paved the way for successful commercial products such as Teradata and Tandem. These systems provide good query performance and low latency but not the scalability required to scale to massive datasets in the face of high application workloads. Parallel DBMSs scale to hundreds of computing nodes; datasets are partitioned across disks on these nodes. Queries are serviced in parallel by dividing the workload into smaller pieces that process data concurrently on multiple nodes with data exchange between nodes over the network. Efficient join algorithms that reduce data communication overheads and techniques that exploit data caching and replication for performance in parallel database systems have been well-studied [62]. Aided by powerful indexing techniques, these systems are optimized for storage of large amounts of data, and the frequent retrieval of subsets of data in response to a range of queries. The database community has also developed federated database technologies to provide unified access to diverse datasets distributed across geographically disparate locations. Spatial databases are extensions to these systems that offer spatial data types (including points, lines and polygons) in their data model and query language, the expression of spatial relationships between these types, a native set of spatial indexing techniques such as the R-tree, and efficient algorithms for spatial joins.

Attempts have been made to develop application-specific support for scientific applications based on top of commercial database systems. However, parallel database
systems have their drawbacks: (i) Traditionally, the relational model has been designed to represent simple entities or objects and the relations between them. For e.g., it may not be possible to suitably represent complex protein structures using available data models without making significant shape approximations. (ii) In general, databases cannot store scientific data in legacy application-specific formats. Instead, data needs to be loaded and reorganized into internal formats specific to the system. (iii) Database systems support querying only using declarative variants of SQL. This query-oriented nature of SQL makes it difficult to express a large number of complex data analysis operations. Moreover, it is not possible to express workflow structure (such as data flow dependencies, or lack thereof, between components) using SQL queries. User-defined stored procedures and functions are supported by some systems, often at the price of reduced query optimizations. (iv) In parallel database systems, the query planner/optimizer and the storage engine are tightly coupled, i.e., the query execution plan generated by the system is resource-dependent. It can be used for execution only by the storage engine that the query optimizer is coupled to.

**NoSQL Data Stores** To address the scalability issues associated with parallel database systems and also to avoid the transactional overheads and consistency overheads (ACID semantics) of conventional database technology, next generation database systems are characterized by several changing trends aimed at improving performance at very large data scales at the expense of true database functionality. There is no longer an adherence to the relational data model – instead these systems are schema-free (i.e., data may not conform to fixed table schemas as in the relational model). As a result, applications need not express analysis requirements through
SQL; user-defined analysis operations can be created and can access the data stores through simple APIs thereby allowing for greater expressivity in the applications. These systems typically employ a distributed architecture where storage and access is governed by a distributed hash table. In this sense, these systems are similar to the peer-to-peer storage systems, and are designed to scale horizontally to thousands of nodes at the web-scale in a manner that is transparent to the applications. NoSQL data stores designed to run on a single large cluster may be built on top of existing distributed file systems (e.g., BigTable builds on Google’s GFS; HBase builds on Hadoop’s HDFS). Such systems could be classified into column-stores, document stores or key-value (tuple) stores depending on how data is laid out and on the interfaces supported to access data within the system.

SciDB is one such NoSQL system designed to address the analysis requirements of petabyte-scale scientific applications. Specifically, SciDB provides a multi-dimensional nested array-based data model for efficient storage and access of scientific data from various domains. SciDB includes support for user-defined functions to carry out customized analysis of the data. Scientific data in legacy formats need not be ‘loaded’ into the system; operations are processed in-situ through domain-specific data adaptors. Flexible data partitioning mechanisms that can change over time allow partitioning based on different dimensions. Queries refer to a single array and a query planner takes care of splitting the query across all the nodes where user-defined functions are run in parallel across the cluster.

**Summary.** In many cases, application instances require access to only small portions of the large datasets. A common approach to improve access to portions of
large multi-dimensional datasets is data chunking\(^{13}\). That is, the contents of the scientific data files are treated as a collection of multi-dimensional arrays and corresponding metadata. Data chunking partitions these arrays into smaller disjoint chunks of user-specified sizes and dimensions. Typically, the datasets are partitioned such that spatially proximate data elements (points or objects) are grouped into the same chunk – an effort to exploit the spatio-temporal locality inherent in many scientific data analysis operations. Indexes are built on data chunks as opposed to individual data elements; this hierarchical indexing scheme helps access mechanisms to scale to very large datasets. As chunks are the unit of disk I/O, system support must be geared towards optimizing the storage and access of chunks. From an application’s perspective, the notion of chunking naturally lends itself to various forms of parallelism in computation – however, existing legacy applications must be modified to operate on out-of-core datasets, i.e., the previously held assumption that entire datasets can be staged into memory is no longer valid for large datasets.

This notion of data partitioning has been successfully applied in the past by file systems (data partitioned and stored as system-level blocks or objects), and by domain-specific libraries such as HDF and middleware such as ADR (data partitioned into application-level chunks)\(^{[71]}\) to optimize storage and retrieval of large datasets in a distributed environment. When data chunking and indexing are carried out at the application-level, the chunks are visible to the application layer. So the question arises as to whether application knowledge (or knowledge within the domain) can be used to configure the chunking (and related operations of indexing, storage and

\(^{13}\)referred to in the rest of this document as just chunking.
retrieval) for improved performance. For example, in the LSST application from use-case scenario 3 in chapter 2, the spatial extent of the observed sky is known \textit{a priori}. This is an example of application-level metadata that could be used to drive data chunking and spatial indexing schemes that are optimal for this application instance.

This chapter presents two proposed approaches for using application-level knowledge within existing systems to improve storage, retrieval and processing of scientific, multi-dimensional datasets at large scales. The first approach targets peer-to-peer style storage systems that are common to global-scale file systems and NoSQL data stores. We refer to this approach as the indirect approach because applications do not explicitly control the placement and movement of data within the system. Instead, applications try to influence the system’s data management policies by scheduling their data access requirements in different ways. The second approach targets active storage nodes within high-end clusters where each node has high-bandwidth access to large amounts of local disk space, i.e., each node mounts an independent file system that allows for parallel data reads off of local disk. Customizable services running on each node form a peer-to-peer network to manage data storage and retrieval for the applications. We refer to this approach as the direct approach because the API allows applications to explicitly control the placement and movement of data within the system.

### 3.2 Multi-dimensional Data Server over a Peer-to-Peer Storage System

In this work, I designed and implemented a distributed data server to access multi-dimensional scientific datasets partitioned and stored in a distributed, peer-to-peer object storage system (called Pond). The main aim of this work was to leverage
the scalable storage and asynchronous replication of data objects in a peer-to-peer storage environment to provide flexible multi-dimensional data analysis services to client applications. Multi-dimensional data services are required for the execution of application-specified range queries into the large datasets. Such queries take the form \( \{D, LB, UB, F, P\} \) where \( D \) is the dataset being queried, \( LB = \{lb_1, lb_2, \ldots, lb_d\} \) is an array containing the lower bounds of the range query along each of the \( d \) dimensions of the dataset, similarly \( UB = \{ub_1, ub_2, \ldots, ub_d\} \) contains the upper bounds along the \( d \) dimensions, \( F \) represents the analysis operation to be applied on the query result by the application, and \( P \) is the input parameter string to \( F \). To answer such queries efficiently, the services need to provide support for partitioning data into chunks, and for efficient storage and indexing of these chunks in the system. The data server must be extensible in the sense that any new user-defined analysis operations must be able to customize and build on these services to provide added functionality.

### 3.2.1 Pond: A Peer-to-peer Storage System Prototype

Pond \[94\] is a prototype of the OceanStore framework\[14\] which is designed to provide support for location-independent access to distributed persistent data in a uniform and universally available fashion. A Pond instance is made up of a number of virtual interactive resources (Pond clients or nodes) that make up an overlay network. Both resources and data are named using 160-bit GUIDs (Globally Unique Identifiers) from the same namespace. The Pond system is built atop the Staged Event-Driven Architecture (SEDA) and uses Tapestry \[115\] as the underlying means of communication between the overlay resources that constitute a Pond instance. Tapestry uses TCP/IP and provides Decentralized Object Location and Routing (DOLR). Pond

\[14\]http://oceanstore.cs.berkeley.edu/
automatically replicates data on or near its resources for greater locality and availability.

The basic unit of data storage in Pond is a data object, identified uniquely by its AGUID (Active Globally Unique Identifier). Pond clients can create data objects, request to read data from the objects, and update the data objects. Objects are automatically and dynamically replicated in the system for greater availability and faster data access. One copy of a data object is identified as its primary copy, while the others are its secondary copies. The consistency of object replicas is also managed by Pond. At the basic level, the actual data is stored in the form of byte arrays in read-only blocks, identified uniquely by their block GUIDs. These blocks of data are cached throughout the system. The main entities present in a Pond system include: (1) Inner Ring (Tier 1), which is a set of powerful and well-connected servers. Any one of the inner ring servers can store the primary copy of a data object, while the other servers automatically become replicas for that object. (2) Pond clients (Tier 2), which are applications that access Pond through local processes. When a new object is created in Pond, it is assigned an Inner Ring server. In order to retrieve a data object, a Pond client must provide a read request containing the following information: (1) the AGUID of the data object, (2) Version Predicate: A mechanism to request specific (possibly older) versions of the object based on its VGUID (Version GUID). (3) Selection: The specific portion of data that needs to be retrieved from within the object.
3.2.2 Server Design and Implementation

Data Storage. In Pond, each chunk of a scientific, multi-dimensional dataset is stored as a data object in the system. The underlying runtime in Pond automatically takes care of distributing and replicating these chunk data objects across the nodes of the system in a manner that ensures high availability of these objects to the applications.

Indexing. We use an Rtree index \[50, 51\] to rapidly search for data that intersects the query bounding box. The index contains the VGUID of the object as part of metadata for the object\(^\text{15}\). This is an optimization to speed up data object retrieval. Every time a Pond node issues a read request for a new object, it needs to provide a version predicate. If it requests the latest version, the request is forwarded all the way to the inner ring for that object. Only when the node knows the VGUID of an object can it issue a read request for the object. Since scientific datasets are usually read-only (updates to data are in the form of adding new data elements to the dataset or new datasets to the system), there exists only a single version of every data object in the system. By including the VGUID of the object as part of the metadata at the time of index creation, the node can know the VGUID by just performing the index lookup, and thus avoid sending the VGUID request to the inner ring. Like data chunks in the dataset, the index is also serialized into a byte array and stored as a Pond object. This makes it possible for one server node to perform an index lookup on the data and answer queries into data created by another server node, thereby potentially reducing the workload on the latter.

\(^{15}\)We should note that more than one R-tree index file can be created if a single index for the dataset will be very big
System Architecture. The data server system is composed of a set of specialized Pond nodes that implement a typical server-client system. These nodes use Pond’s rich interfaces to create, store, and read data (objects). Based on their functions, these nodes are classified as one of the following. (i) Master Server: A master server is a server that creates Pond objects in the system. Master servers have multi-dimensional datasets stored on local disks. On entering the system, they partition input datasets into data chunks and create Pond objects for each of these chunks. They also create indexes for efficient retrieval of these objects. The object creation phase is a one-time operation for a master server because objects in Pond remain forever, as long as the Pond instance is up and running. Once a master server is done creating data, it functions as a normal server that receives requests from applications, performs index lookup, retrieves data, and sends the results back to the application client. A master server can also redirect some of its queries to other servers (called slave servers) if it becomes overloaded. (ii) Slave Server: As the name suggests, the slave server acts as a slave to master servers. Excess queries can be redirected to slave servers to ease the burden on the master. A slave server can receive queries directly from a client. It can also redirect queries to other slave servers. The presence of many slave servers can also lead to greater caching and replication of data. (iii) Resource Manager: In some of the query execution strategies, we employ a centralized entity called a resource manager, which keeps track of the number of servers in the system and their loads. Servers and clients contact the resource manager to find out the least loaded server and redirect their queries to that server. (iv) Client: A client is an application that
sends queries to the servers requesting data. Multiple query requests may be issued by an application during the course of its execution.\footnote{In our implementation, clients do not provide disk storage for the data blocks distributed throughout the system.}

**Query Execution.** Once a server receives a query, it looks for a copy of the dataset object $D$ on local disk. If it is not present, the server retrieves the corresponding object from Pond and stores it locally. The dataset object contains the list of indexes (i.e., index objects/files) associated with the dataset. The server then retrieves the Pond objects corresponding to those indices and stores them on local disk as index files for the R-tree index \[51\]. This is followed by an index lookup on each of these indices. The objects corresponding to the chunks intersecting the query are retrieved from Pond by the server, and results are sent back to the client using the Tapestry \[115\] layer in Pond, i.e., result data is routed back to the client as a set of chunks via Tapestry’s routing support.

**Analysis operations.** User-defined operations such as value-based filtering and data aggregation may be specified as part of the query submitted against a scientific dataset. Such analysis operations need to be performed by the data server system on the retrieved chunks before routing the results back to the requesting client. User-defined operations are compiled as a Java class and stored as Pond objects. This allows any server in the system to be able to execute queries with user-defined processing operations. In our implementation, each user-defined operation is assigned a unique name and the AGUID of the operation is derived from its name by using the hash function employed in Pond. When a query is received by a server, the server retrieves the corresponding Pond object, extracts the Java class, and creates a Java object using the `ClassLoader` object provided by Java. A user-defined operation is
implemented as an object with well-defined callback methods in the data server. Currently, the object is expressed using a Java language binding by subclassing a base class. Drawing from the notion of filters and aggregation objects in [23], the interface of method callbacks consists of an initialization function, a processing function, and a finalization function. The initialization function is called when the query is received to initialize internal data structures. The process function is executed for each data chunk that intersects the query bounding box. The finalize function is called after the last data chunk is processed.

3.2.3 Query Scheduling and Performance Optimizations

Since any node can perform an index lookup and retrieve data in Pond, client applications can send their multi-dimensional queries to any of these nodes. We have investigated several strategies, one static and two dynamic scheduling schemes, in our implementation. The objective of dynamic query scheduling strategies is to route the queries to appropriate nodes in the system so that the workloads on the nodes are evenly balanced at all times and the queries are answered as quickly as possible. The first dynamic strategy uses a centralized server-side entity, while the second dynamic strategy adopts a client-side distributed approach.

**Static Partitioning (Static):** This strategy evenly distributes clients among servers (both master and slave servers). Each client is assigned to a server statically and submits all its queries to that server. Each query is answered by the server to which the client is assigned.

**Resource Manager:** Every server, master and slave, registers itself with the resource manager. The resource manager keeps track of the load on each server. We
have implemented two variations of this strategy. In the *Resource Manager Server Only* (RMS) strategy, clients direct all their queries to any one server. When a server receives a query, it notifies the resource manager so that the resource manager increments the count of queries being handled by the server. If the server detects that the number of queries exceeds the excess load threshold, the server sends a request to the resource manager for the least loaded server. Once the least loaded server is known, the server forwards the excess queries to it. It then notifies the resource manager to decrement the count of queries assigned to this server accordingly. In the *Resource Manager Client-Server* (RMS+RMC) strategy, a client first contacts the resource manager to find out the least loaded server and sends its queries to that server. Once the query is received by a server, the RMS strategy is employed, if a server becomes overloaded. The goal of using the RMS strategy in addition to client’s contacting the resource manager is to allow servers to dynamically change their threshold for excess load or apply a different method to compute their load (instead of just counting the number of queries waiting in the queued).

**Client-side Dynamic Selection (Client-Dynamic):** The previous strategies use the number of queries assigned to a server as the criterion for choosing a server to delegate queries. However, that may not always yield a good result. For instance, if the server chosen is far away from the client, the advantage gained by servicing the queries as early as possible is lost due to the time it takes to route the data back to the client across a large distance. The client-dynamic strategy uses the query response time as the criterion for choosing who executes queries. The client initially sends a small ‘test’ query to each server. It records the response times from each server. The next query is sent to the server that provides best results for this client.
The client updates its records every time it receives a response. If a previously ‘fast’ server yields poor response times, then this may be reflective of the increased load on the server, or increased traffic in the network. The client-side dynamic selection strategy is most beneficial when a client will submit many queries to the data server system over the client’s lifetime or when there is a proxy located near a group of clients, which handles submission of queries to the system and routing of results back to clients on behalf of the group of clients.

Another client-side strategy we implemented is the Random Server Selection strategy. In this strategy, the client chooses a server at random and sends its query to that server. This is a simple strategy that does not require keeping track of response times or the number of queries assigned to servers. By randomizing selection of servers, this strategy tries to balance distribution of queries to servers and at the same time, to reduce the chances of two clients submitting queries to the same server concurrently.

### 3.2.4 Experimental Evaluation

For the experimental evaluation, we used a heterogeneous platform consisting of two inter-connected clusters. The first cluster, OSUMED, is made up of 24 Linux PCs. Each node has a PIII 933MHz CPU, 512 MB main memory, and three 100 GB IDE disks. The nodes in this cluster are inter-connected via Switched Fast Ethernet. The second cluster, MOB, contains 8 nodes. Each node of the cluster consists of two AMD Opteron processors with 8 GB of memory, 1.5 TB of disk storage in RAID 5 SATA disk arrays. The nodes are connected to each other via a Gigabit switch. MOB is connected to OSUMED over a shared 100 Mbit network. For our experiments, master servers and inner ring servers were run on nodes in the
MOB cluster. Slave servers and clients were executed on nodes in the OSUMED cluster. We ran experiments to evaluate the performance of the various scheduling strategies. Our main objective is to reduce the overall time taken to serve queries from all clients in the system. In the experiments, the bounding boxes of input data chunks are randomly created using uniform distribution within a 2-D \([0,1] \times [0,1]\) rectangle and a client requests subsets of chunks via 2-D queries.

![Dynamic selection of servers by clients](image)

**Figure 3.1:** Performance comparison of the client-dynamic strategy vs. random assignment of queries to servers

The first set of experiments compares the performance of the client-dynamic strategy to that of the client-side random server selection strategy. The experimental setup consists of a master server that creates 120 MB of data, six slave servers, and a single client that sends thirty randomly-generated queries. In the client-dynamic strategy, the client first sends test queries, each of which returns a single data chunk, to all the servers in the system, and uses the observed bandwidth to rank the servers initially. In our experiments, the size of the data chunk returned as response to a test
query is 512KB. The timing results in the graphs do not include the time for the test queries, which are submitted only once when a client joins the system. In the current implementation, the client waits for all responses to come back, so that it can build its local table of response times for each server. We observed that the overhead from a test query is about 6 seconds in our experimental configuration. Figure 3.1 gives us the breakup of times after every 5 queries are executed. We observed that random assignment achieved a more balanced distribution of load in terms of the number of queries assigned to servers. However, in terms of response time, the client-dynamic strategy performs better since it takes into account bandwidth between the client and a server.

Figure 3.2: Performance comparison of query scheduling strategies

The second set of experiments compare the performance of different query scheduling strategies. In these experiments, there is one master server, 6 slave servers, and 12 clients. Each client submits a total of 10 randomly generated queries one by one to
the system. The master server stores 250 data chunks, each of which is 512KB in size, in Pond. Figure 3.2 shows the maximum and minimum response times observed by the clients when 10 queries from each client are answered. As is seen from the figure, the client-dynamic strategy achieves a better performance among all the strategies. The static strategy aims to achieve load balance across servers by evenly partitioning clients among servers. The RMS and RMS+RMC targets load balance at the query level and assigns queries to least loaded servers. The client-side client-dynamic strategy, on the other hand, aims to minimize query response time as observed by the client. Our results show that the metric used by the client-dynamic strategy is a better approximation of load distribution and network overhead. Figure 3.3 shows the maximum and minimum number of queries handled by servers in the system. In the dynamic strategies, unlike the static strategy or the random server selection strategy, a particular server may yield fast query execution times and response times for the client, and as a result, may have more queries directed toward it.

![Load balance among servers](image)

Figure 3.3: Distribution of queries across servers with different scheduling strategies.
In order to look at the variation in execution time when the number of clients is varied, we carried out a set of experiments using a fixed number of 6 servers. The number of clients was 3, 6, 9, and 12. Figure 3.4 shows the total execution times for different number of clients. In these experiments, we used the client-dynamic strategy. The execution times in the graph are the response times observed by the client for 10 queries. \textit{Fastest}, \textit{Average}, and \textit{Slowest} denote the minimum, average, and maximum response times among clients. As expected the execution time rises as the number of clients is increased, since more clients have to be served by the given set of servers. We observe that the increase in the execution time is linearly proportional to the number of clients.

We also conducted a set of experiments to measure the relative performance of the various strategies for slicing queries using the same configurations as in our original experiments. Randomly generated slicing queries that extracted row-wise and
column-wise thin slices were submitted to the system by 12 clients. Each client submitted 10 queries. We observed performance trends between algorithms which were similar to those in our original range query experiments.

3.2.5 Conclusions

This section presented an approach to designing an extensible multi-dimensional data server atop a peer-to-peer storage system. The benefits of such a data server will manifest themselves when a large number of client queries are issued concurrently against large datasets. We implemented and evaluated different application-level optimizations to improve query execution times: (i) **Structural optimizations** take advantage of the fact that no restriction is placed on what data a Pond object may contain, so long as it is in serializable form. In our application, in addition to storing the data chunks as Pond objects, the index into the dataset is itself serialized into an array of bytes and stored as a Pond object. Due to replication, other nodes in the system too have easy access to this index object. So, it is possible for one node to perform an index lookup on the data and answer queries into the data created by another node, thereby reducing the workload on the latter. Also, the index is not stored at a single location, and so there is no single point of failure. (ii) **Functional optimizations** are optimizations of functionality specific to the application at hand. Since any node can perform an index lookup and retrieve data in Pond, client applications can send their multi-dimensional queries to any of node. Our strategies help to route the queries to appropriate nodes in the system so that the workloads on the nodes are evenly balanced at all times and the queries are answered as efficiently as possible. (iii) **Data Processing Optimizations**: Multi-dimensional data servers can do
more than just service data subsetting queries. In many cases, the data of interest may span only a portion of single/multiple chunks. The server is expected to clip the unwanted data and send only the relevant data back to the client. Also, a lot of image datasets contain multi-resolution data. Servers are expected to perform super-sampling(sub-sampling) operations on the data to service queries into data at high(low) resolution. In our application, we implement these special operations in a generic fashion, and store their executable forms as Pond objects. So, it is possible for any node in the system that has sufficient computing resources to execute these operations on the data.

Based on preliminary results gathered on the system, we believe that different query scheduling strategies can be employed to balance the load amongst the system’s resources. The storage system is completely responsible for placing and replicating data objects. However, through strategies such as Client-dynamic, the client application is letting the server know the regions of the dataset that are of high interest. This leads to asynchronous replication of relevant data objects on nodes along the path from the server to the client. In the long run, this indirect passage of an application’s data access patterns to the server leads to greater availability of requested data in the system. As a result, improved performance per query is likely to ensue.

3.3 Mediator Services for I/O Virtualization

In this work, I designed a layer of multi-dimensional data services, collectively referred to as mediator services, towards chunk-level data management on storage nodes
within high-end cluster environments. The main aim of this work was to provide parallel application programs with a simple, high-level API and a customizable set of services to enable them to retrieve and process out-of-core scientific, multi-dimensional data in a scalable and efficient manner. Compared to the system presented in section 3.2, the mediator services can be considered to be similar in functionality to the master and slave servers in the inner ring. A crucial difference in this work is that we want to provide the application programs with the ability to explicitly manage the storage and retrieval of data in the system for optimal performance. That is, this is a no-replication setting, where we seek absolute performance at the expense of fault tolerance and high availability. By circumventing existing distributed file systems, analysis must be carried out on out-of-core data distributed and stored on local disks of the storage nodes. This has some potential advantages: (i) the overheads associated with querying file system metadata servers repeatedly is avoided, and (ii) through judicious data partitioning, one can have a situation where each storage node needs to read data only from its local disk(s) – not only does this avoid the slow network overheads of file systems, but also exploits high-performance parallel I/O access to local disk on each node. Our motivations are shared by developers of systems like Zazen [106] where postprocessing of scientific simulation data is carried out on a set of nodes accessing local disks.

Applications need support for out-of-core data analysis as datasets could potentially be larger than the collective memory of a cluster. When data chunking is used and each chunk is small enough to fit in memory, applications need to be modified to operate on multiple chunks instead of entire datasets. For complex analysis operations, this could result in repeated accesses being made (via range queries) to several...
portions of large datasets. Improving data storage and retrieval for efficient processing requires application-driven mechanisms for caching and prefetching relevant data chunks. To minimize disk I/O overheads and optimize disk bandwidth, the number of times a chunk is read off or written to storage must be minimized.

3.3.1 Mediator Design

Mediator services are a collection of configurable data services that assist application programs to explicitly dictate the data placement and movement within the system when operating on out-of-core data. The different configurations are aimed at performance optimizations when managing large multi-dimensional datasets.

Data Partitioning and Indexing Services: Multi-dimensional datasets are initially partitioned into chunks of regular shapes and sizes. The application can specify the partitioning strategy to be employed, and the dimensions and sizes of the chunks using high-level directives such as “Uniform partitioning into 2D chunks of size 500 × 500 data points along X and Y dimensions”. In addition, the application can specify different degrees of overlap between non-disjoint chunks along different dimensions. Alternatively, one can specify a non-uniform partitioning strategy such as Recursive Bisection for certain datasets. A data partitioner service then performs the necessary low-level implementations to partition the dataset, and then generates and maintains the application-level metadata details about the chunks and any indexes created to access the chunks. The partitioner service also drives the storing of the chunks on local disk on the cluster nodes in either a raw form or in a compressed form using compression techniques suitable to the domain under question (e.g., chunks of image regions could be compressed and stored in the system in the form TIFF files).
**Data Distribution Service:** This service allows the partitioned data chunks to be distributed among the nodes of the cluster in application-specified ways. Static distribution strategies are supported for applications that analyze offline data. For applications that processing streaming data, this service supports an on-demand dynamic distribution strategy that takes into account the balance of workload between the nodes (processing workloads may differ from one chunk to the other). Scalable storage systems commonly support hash-based distribution strategies where data is distributed based on a hash of one of the chunk-level attributes. However, in our system, a variety of distribution strategies are supported in order to exploit any inherent spatio-temporal locality in the multi-dimensional data analysis operations. In addition to hash-based distribution, this service provides low-level support for round-robin, stacked and Hilbert-curve based distributions which may yield better performance than the former. That is, when certain applications access a given chunk, the spatial locality of access may suggest that spatially proximate chunks will be accessed by the same application in the near future. In such cases, strategies that map spatially proximate chunks to the same set of nodes may be preferred. Another feature of the static distribution strategies is that they model the relative load on each storage node – if a node has limited storage capacity or is perceived to have low disk bandwidth, then it is assigned a lower weight as compared to other nodes; nodes with lower weights have lesser number of data chunks mapped to them for analysis.

**System Architecture:** Our system is composed of a set of common services that support application processes in reading and writing data chunks in a location-independent manner. Application programs need not be aware of the exact physical location of chunks in the system. Instead, they rely on a set of *mediators* that
provide I/O virtualization by abstracting details about the underlying storage systems on different nodes, and present the applications with a simple, unified API to read and write multi-dimensional data chunks in different formats. The mediators are responsible for chunk management, and assist the applications in determining (i) on which node a requested chunk is located, (ii) where on local disk on a node can a requested chunk be accessed, (iii) on which node should an output chunk be written?, and (iv) how should an output chunk be stored on disk on a node? The mediators provide a common mechanism to read a chunk regardless of whether it resides on local or on remote disk. It also provides a means to write out a new chunk onto local or remote disk for data products currently being created by an application instance, and a mechanism to finalize all written chunks at once into their final output destinations, such that total success or total failure in writing of all chunks transpires (atomicity of writes). Figure 3.3 shows an instance of mediator services set up across three sets of cluster nodes. Application programs executing on the compute cluster may need to access and store data onto the storage system (which in this case are local disks on a set of active storage nodes in the cluster). Mediator instances are invoked by the system on every node. These mediators communicate in the form of a peer-to-peer network of processes, exchanging application-level metadata and chunk read/write requests and responses among themselves.

**Query Execution:** A mediator instance receives chunk read/write requests from application processes and works with mediator instances on other nodes to carry out the requests. That is, if a mediator receives a request from an application or

---

17 Data chunks may also be stored on local disks of the compute nodes if they have adequate disk capacity and bandwidth to be considered a part of the storage system. Likewise, application program instances can also execute on the active storage nodes.
Figure 3.5: Mediator services instantiated on each nodes to help applications access disk-based chunks
from another mediator to read/write a chunk on its local disk, then it performs the necessary I/O operations. If it is a request to remote disk, i.e., the requested chunks or chunk locations lie on other nodes, then it passes on the requests to the mediator instances running on those nodes. Once the application programs are done analyzing all the data and issue no more requests for chunks, then mediators complete any pending chunk writes, finalize and exit. Figure 3.6 show how the actual I/O to

![Diagram](image.png)

**Figure 3.6: Common set of services for transparent I/O to local disk on a node**

a local file system is, in turn, delegated by a mediator instance onto a number of *mediator_reader* and *mediator_writer* services on the same node. The number of mediator_reader and mediator_writer instances per node can be tuned based on the relative data read and write bandwidths of the storage system on the node. The *rangefetcher* service is used to hide data retrieval latency. It issues a series of requests
on behalf of the application to fetch a number of chunks in a sequence if these chunks are known ahead of time. This way, the rangefetcher can work slightly ahead of the application, minimizing chunk retrieval (either from local disk or from a remote node) latency for the application. That is, when an application is working on chunk $C_1$, the rangefetcher filter is reading chunk $C_2$, such that when the application needs $C_2$, it is ready and waiting in memory, thereby reducing read latency. If a multi-dimensional range query spans across multiple chunks, then the indexing service is looked up, and a read request is issued to the mediator for every chunk that intersects with the query region.

### 3.3.2 Implementation: Out-of-core Virtual Microscope

Application programs that issue chunk read requests typically perform user-defined processing operations on the retrieved chunks, ahead of issuing write requests for the processed chunks. These potentially parallel programs could range from simple value-based filtering operations to complex analysis tasks such as image warping that are specific to a domain. User-defined analysis operations can be expressed in high-level languages and encapsulated within components that have standard interfaces. To experimentally evaluate the mediator layer, I designed and implemented a data analysis framework called the Out-of-Core Virtual Microscope (OCVM), to enable legacy microscopy image analysis codes to scale to multi-terabyte scale images using the support of the mediators. OCVM has been developed as an extension to the Virtual Microscope tool to support virtual microscopy operations on large image datasets.

While the implementation of OCVM is specific to the management of multi-dimensional image data, one could extend the basic mediator-based framework to support data analysis operations on generic multi-dimensional datasets.
OCVM is designed to be able to execute coarse-grain workflows of parallel data analysis operations on large clusters. To enable this, OCVM builds on existing component-based programming frameworks and makes them scale to massive data volumes by providing specialized components that implement the aforesaid mediator functionality. Workflow applications developed using these component frameworks can invoke instances of the specialized mediator components for transparent access to a large virtualized storage system. For our implementation, we developed instances of OCVM over two popular component frameworks: the DataCutter middleware \cite{datacutter} developed at the University of Maryland and the Ohio State University, and IBM’s InfoSphere Streams \cite{infosphere} system. These systems provide a stream-based programming model where components (referred to as ‘filters’ or ‘operators’) exchange data using a stream abstraction to provide pipelined dataflow execution on cluster nodes. The filter-streaming model is a good choice for large-scale data analysis applications because it naturally exposes several forms of parallelism to the developer.

DataCutter provides a core set of services, on top of which application-specific services can be implemented. One of these core services is the filtering service for executing application-specific processes as a set of components, called filters, in a distributed environment. For each component, the analysis logic (expressed using high-level languages like C++, Java, MATLAB and Python) is embedded into one or more filters in DataCutter. Users describe an application workflow in the form of a graph where each vertex represents a filter and the edges represent the data flow dependencies between the filters. Users also provide a list of cluster nodes across which to carry out the execution. All filters in DataCutter are bound to compute nodes at runtime. The filtering service within DataCutter performs all steps necessary to
instantiate filters on the target nodes and invokes each filter’s analysis logic. Each filter executes within a separate thread, allowing for CPU, I/O and communication overlap during the analysis. Multiple copies of a filter can be created for data parallelism within a component. The filters communicate and exchange data via a stream abstraction. A stream denotes a unidirectional data flow from one filter (i.e., the producer) to another (i.e., the consumer). Data exchange among filters on the same node is accomplished via pointer hand-off while message passing is used for filters on different nodes. The version of DataCutter employed in our work uses flavors of MPI for communication in order to exploit high-speed interconnect technology on today’s clusters. For each component of the workflow, application-specific filters are created that implement the processing of data chunks. Each such filter has a well-defined interface for invocation and usually operates on a set of one or more input data instances. Application filters, like all other filters in DataCutter, are location-independent and can be bound to any node at runtime. The user can optionally also provide metadata information about input datasets (in case this information is not explicitly available within headers of the data elements).

Traditionally, filter-stream programming frameworks were designed to perform light-weight analysis tasks, where analysis can be performed on the data at the rate at which it streams through the filter. However, in many complex analysis tasks in our target applications, we may have heavy-weight operations that may not be able to process data in one pass. Moreover, since a reduction in data is not guaranteed as a result of processing, these frameworks run the risk of exhausting the physical memory on a node especially when multiple filter instances are co-located on the same node. OCVM is implemented as an extension to DataCutter (or Streams) where each
instance of the mediator service and the partitioning and distribution services are implemented as specialized DataCutter filters (or Streams operators). The \texttt{DCMediator} filter in DataCutter extends the base \texttt{DCFilter} class and implements additional functions that application filters can use to read and write data chunks. Application filters that need to analyze out-of-core multi-dimensional data now extend the \texttt{DCMediator} class to avail these functions. Those filters that do not need support for out-of-core data management extend the \texttt{DCFilter} class as was done previously. With the added support from OCVM, the DataCutter runtime now also instantiates mediator filters on each node within the system and sets up the stream connections between mediator filter instances and also between application filters and their corresponding mediators on each node. Figure 3.7 shows the layout of filters within an instance of OCVM used to carry out a user-defined analysis operation called ‘normalization’ on the data. In the figure, each oval represents a running filter on a compute node in the system, and each arrow represents a communication line between two filters. In this instance, we have a single normalization filter per node that relies on the mediator and range-fetcher services on the respective node to carry out chunk read requests. This layout of filters could be a part of a much larger workflow containing application filters for other analysis stages.

Along with mediator services for distributed chunk I/O, OCVM also extends the basic filter-stream programming framework with supplementary analysis filters aimed at easing the development of microscopy image analysis code. In general, the application filter code in DataCutter can be expressed in C++, Java or Python. However, for many image analysis applications, MATLAB is the language of choice among scientists, as it allows rapid prototyping and provides rich a rich set of libraries. So,
Figure 3.7: Filter layout for Normalization operation in DataCutter with mediator support from OCVM
for MATLAB applications, OCVM includes a supplementary filter known as a *matlab_invoker* that invokes and runs an instance of MATLAB in the background on the node, waiting for commands. The application filter then submits commands like a user normally would to the *matlab_invoker*. Similarly, when object-relational data are analyzed, they are usually stored and queried using database systems like MySQL or PostgreSQL. OCVM provides a supplementary filter, *db_invoker* that starts up the database server daemon in the background on a node. The application filter can then store data tuples on the node and submit SQL queries to the *db_invoker* filter just as a database client would normally submit queries to a server. Finally, microscopy image data obtained from the instruments is often stored in proprietary image formats developed by the vendors. OCVM provides supplementary filters known as *edge adapters* that can automatically transform image data in such standard formats as TIFF and Aperio’s SVS into data chunks for use by the application filters.

### 3.3.3 Experimental Evaluation

This section presents a basic experimental evaluation of OCVM and demonstrates its benefits to applications through a use-case scenario. For this evaluation, our testbed consists of two Linux clusters located at different departments (about a half a mile apart) within the Ohio State University campus. These clusters were set up as part of an NSF-funded research infrastructure grant. Each cluster is representative of a high-performance Grid site, and consists of 64 high-end computing nodes. One cluster (referred to as **RII-MEMORY**) has dual Opteron 250 (2.4 GHz) processors equipped with 8 GB of memory and interconnected by a switched 1Gbps Ethernet network. The storage system on each of these nodes consists of 2x250GB SATA disks
installed locally, and joined into a 437GB RAID0 volume with a RAID block size of 256KB. The maximum disk bandwidth available per node is around 35 MB/sec for sequential reads and 55 MB/sec for sequential writes. The second cluster, (called RII-COMPUTE), is a heterogeneous cluster consisting of a mixture of dual Opteron 254 (2.7 GHz) nodes, each with 4 GB of memory and 10 GB of local disk space, and dual Xeon (3.2 GHz) processors with 2 GB of memory and 10 GB of local disk space. These nodes are interconnected by both an Infiniband and a switched 1Gbps Ethernet network. The RII-MEMORY and RII-COMPUTE clusters are connected using 10-Gigabit wide-area network connection – each node is connected to the network via a Gigabit card; an 8 Gbps application-level aggregate bandwidth was observed between the two clusters.

We ran experiments to evaluate the scalability of the mediator services in supporting different analysis operations on microscopy image datasets. Each image is uniformly partitioned into 2D chunks of size $512 \times 480$ pixels and distributed across the nodes of the RII-MEMORY cluster. The user-defined analysis operations used in our evaluation are low-level data transformation tasks borrowed from the PIQ workflow (use-case scenario 1 in chapter [2]) and include zproject, normalize, autoalign and stitch. Parallel implementations of these operations execute on upto 16 compute nodes of our testbed. Each node hosts one or more instances of filters corresponding to each application component. We evaluated the scalability when using two different execution configurations – local and remote. In the local configuration, application filters are instantiated on the same nodes that host the input data (the nodes of the RII-MEMORY cluster), and hence are ‘local to the data’ being processed. Any intermediate and output data are also written onto disks on the same set of nodes.
that store the input data. Thus, any data exchange between mediators is restricted to a set of nodes within the RII-MEMORY cluster.

Figure 3.8 shows scalability with number of nodes, when processing a 6.4 GB image (49152x46080x1 pixels). We observe that as the number of compute nodes is doubled in the range of 1 through 16, the total runtime is roughly cut in half, thus almost linear speedup is achieved for each of these components. Figure 3.9 shows scalability with data size. We used input images with different sizes (1.5GB, 6GB, 25GB, 100GB and 400GB) and carried out the execution on 16 nodes. As the image size is gradually increased by a factor of 4, we should expect the runtime to increase by a factor of 4, if we are scaling linearly. Our results confirm linear scalability,
since the total runtime increases by no more than a factor of 4 with each dataset size increase.

![Figure 3.9: Scaling data size for zproject, normalize, autoalign and stitch operations.](image)

In the remote configuration, all data to be processed by the application filters reside on remote disks (and in this case, on nodes of a different cluster altogether). All intermediate and output data from the application filters running on the RII-COMPUTE cluster must be written back to disks on the RII-MEMORY cluster. To facilitate this, mediator filters are instantiated on all compute nodes that host application filters, and on all storage nodes that house the required data. The mediators
can be expected to exchange large amounts of data over the network. For remote configuration evaluation, our input dataset was a 5 GB image (15360x14400x8 pixels) dataset distributed across the nodes of the RII-MEMORY cluster. We observe that

Figure 3.10: Scaling # of Compute Nodes for zproject, normalize, autoalign and stitch operations. Remote configuration

as the number of compute nodes in each cluster is increased in the range of 1 through 8 (thereby increasing the number of mediator filters in the system from 2 through 16), the execution time for each operation is roughly cut in half, thus almost linear speedup is achieved for each of these components.

Depending on the chunk size and dimensions chosen, the number of chunks stored in the system will vary. Consequently, the number of read/write requests made by

\(^{19}\)In this experiment, the number of compute nodes with application filters was set to be equal to the number of storage nodes across which input data is declustered. In general, this need not be the case.
the application filters (and the workload on the mediator services) will increase or
decrease as the case may be. Similar scalability trends as above were observed even as
we varied the sizes of the chunks into which the input image data is partitioned. We
were also able to demonstrate scalability at higher data scales when we executed all
analysis operations on a 1-Terabyte sized image using 32 nodes of RII-MEMORY [69]
cluster (for the local configuration), and on a 0.5-Terabyte size image using 64 nodes in
our system (for the remote configuration; 32 nodes of the RII-COMPUTE cluster host
the application filters and 32 nodes of the RII-MEMORY cluster store all data) [69].

3.3.4 Performance Optimizations

Our experimental evaluation so far shows that mediator services can govern the
access to chunks on local disks of nodes on a cluster and help parallel application pro-
grams to linearly scale to process multi terabyte-sized datasets. Our goal is to have
the mediator layer not just provide scalable I/O virtualization but also to act as a con-
duit for application- and domain-specific knowledge to be explicitly conveyed by the
application to the system. This section describes the configurability of these services
and shows how they can be customized for performance depending on the applica-
tion characteristics. This involves identifying opportunities for optimization within
the application, and mapping such opportunities onto appropriate configurations of
the mediator services. We illustrate support for different optimizations using a set
of data preprocessing analysis operations borrowed from the PIQ workflow. Prepro-
cessing consists of thresholding, tessellation and 2D prefix sum generation operations

20The largest real image we obtained from a confocal microscope had an uncompressed data size of
about 63 Gigabytes. To show scalability in our results, we scaled up these real images artificially by
varying factors upto multi-terabyte sized images. We did so by generating replicas of these smaller
images, reproducing tiles to the right, below and the lower right (this enabled an easy scale-up by a
factor of 4).
performed on input image data as described in section 2.2.1. At large data scales, these operations presents optimization opportunities that we believe are common to several multi-dimensional data analysis applications that share similar requirements.

Our implementation has one mediator filter per node that stores input data chunks on local disk, and one or more filter instances for each operation per node. The thresholding step, realized through one or more *thresholder filters* per node, reads disk-based input image data chunks (via mediators filters) and simply overwrites each pixel’s color value (Red, Green, Blue) with a 0 or 1, depending on whether the value is outside or inside a user-defined threshold. A different threshold can be specified for each color (R, G, B). The tessellation step, realized through one or more *tessellate filters* per node reads thresholded data chunks as input and produces tessellated chunks, which consist of the sums of all non-zero pixels within user-defined tessellation regions or cells. The tessellation region is specified by the user in terms of an $x$ and $y$ range, and $x \times y$ becomes the factor by which the image resolution (the size of each chunk) is reduced. The prefix sum step, realized through one or more *prefix sum filters* per node, reads tessellated data chunks and computes the 2D prefix sum of the values at each point of the tessellated image and stores the resultant chunks of prefix sums on disk (via mediator filters) for future querying.

Filter-streaming frameworks like DataCutter provide support for combined use of different forms of parallelism to optimize computational performance. Filter execution is concurrent by definition; hence, filter instances processing different data chunks on each node provides data parallelism. Additionally, there is support for task parallelism where multiple filter instances concurrently perform different processing operations on different chunks of the same dataset. Finally, since each filter instance executes
as a thread, transparent copies can be created for each filter instance at runtime to
enable bottleneck operations (like prefix sum generation) to keep up with increases
in load, or to exploit multi-core processing capabilities on the node.

Performance optimizations brought about by explicit configuration of the mediator
services are described below:

**Data distribution:** Figure 3.11 shows some strategies based on which the data
distribution service can map data chunks to nodes. If configured for a *stacked* distri-
bution, chunks are distributed in rows across each node, placing all of a node’s rows
on the node before moving onto the next one. In the *stacked round-robin* configura-
tion (Stacked R-R), rows of chunks are mapped onto nodes in a round-robin fashion,
one row at a time. In configured for a *random* distribution, chunks are distributed at
random among the nodes.

<table>
<thead>
<tr>
<th>Stacked</th>
<th>Stacked R-R</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 1 1 1</td>
<td>1 1 0 1</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>0 0 0 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1 1 1 1</td>
<td>0 0 1 1</td>
</tr>
</tbody>
</table>

Figure 3.11: Data distribution strategies (The presence of a number in a chunk loca-
tion indicates that the node identified by that number contains that chunk)

There is a one-to-one mapping between the number and dimensions of input data
chunks, thresholded chunks, tessellated chunks and prefix sum chunks. However,
the data volumes of chunks produced by each operation may vary. Strategies for
distribution of data chunks among nodes may not affect data parallel operations
like thresholding and tessellation. But with prefix sum generation, the content of
all but one chunks in the dataset depends on one or more other chunks. So, data
distribution can influence the frequency of stalls in waiting for these dependencies to be resolved in communication-intensive operations like prefix sum generation. The communication patterns in an operation like prefix sum generation that produce a global data structure as output involve data exchange between each node and the nodes that store the chunk’s immediate neighbors. To minimize the network communication overheads, we configured the data distribution service to adopt a \textbf{stacked} distribution strategy for our image data chunks.

The preprocessing operations of the PIQ workflow need to be carried out on the dataset each time the end-user modifies the threshold value for any color dimension. Since, as the name suggests, these are merely preprocessing operations that generate some global data structure (like 2D prefix sum or multi-dimensional histogram) to be used for future time-critical querying or monitoring, it is important to minimize all overheads for these operations to obtain optimal performance at large data scales. Optimal utilization of the disk bandwidth on a system can be achieved by configuring the number of \texttt{mediator\_reader} and \texttt{mediator\_writer} filter instances on each node. We used two \texttt{mediator\_readers} per node so that chunk reading could be done in parallel. We used one \texttt{mediator\_writer} per node because the write bandwidth to local disk on our nodes was high enough to keep up with production rate of output chunks by the prefix sum operation. Our next set of optimizations target the communication overhead incurred when data chunks need to be exchanged between filter instances corresponding to the different preprocessing operations.

**Data partitioning:** Co-location of filters can be used to minimize network communication. That is, in case the threshold, tessellate and prefix sum filter instances
that operate on the same data chunks are all located on the same node, then communication is equivalent to handing off a pointer, in other words, nearly free. This cheap communication is made possible because every filter is running as a thread sharing a global address space. On the other hand, filters on different nodes communicate over the network and are subject to network contention. In our evaluation, we saw as much as a 12x improvement for the filter co-location case due to minimized network overheads as opposed to a round-robin assignment of filter instances to nodes. However, filter co-location comes at the price of increased memory consumption per node, since all filter instances now share the available memory on a node. In addition to the input data chunk which must reside in memory, each filter instance could generate intermediate data chunks that must also be allocated memory during execution. It may not possible to maintain all the data chunks from co-located filter instances in memory at once.

When the goal is optimize network bandwidth, one option to address the memory constraint is to work with smaller data chunks. The mediator service provides a runtime optimization wherein such a chunk partitioning can be carried out dynamically if memory is insufficient – the sizes of the intermediate data chunks produced by the filter instances are repeatedly divided in half until all chunks can be expected to fit in memory. Every time a filter instance populates data output of the newly determined chunk size, the data is staged out and written by the mediator to disk as an output chunk (Figure 3.12). Since chunk data is typically stored in row-major order, the divisions are performed horizontally so that the seek overheads within a chunk are minimized. Such a runtime optimization would be possible only in cases where the original data partitioning results in chunks that have a regular shape and dimensions.
Figure 3.12: Dividing chunks horizontally at runtime

Regular rectilinear chunk partitioning can also be used for data caching as will be shown in chapter [4].

**On-disk chunk layout:** A typically layout of chunk data on disk on a node has the X-dimension as the leading dimension, followed by the Y and Z dimensions to benefit row-major access. But given the multi-dimensional nature of the data, certain analysis operations may need to process data values only along a specific subset of dimensions. For example, if the user modifies the threshold only for the red color channel in the thresholding operation, then the threshold filters need to process only the red color values for the pixels within a chunk. As a chunk is the unit of I/O, the entire chunk will be read off disk via the mediator. However, data laid out in a suitable way within a chunk may potentially lend itself to hardware-based acceleration of the computation. The obvious algorithm to perform the thresholding is to look at every byte one at a time and set the byte to 0 or 1. However, with a long sequence of bytes, loop overhead becomes significant, as revealed through profiling. This overhead can be reduced by using vector operations to perform the threshold. This optimization can be performed on any 8-byte aligned 8-byte sequence (of which the thresholding filter processes many), when Intel MMX with SSE extensions is
available (Pentium III and above) [54], and the threshold value given for a color is <= 254. We enable our vector operations at compile time if the architecture supports it. In our implementation, vector optimizations require g++ 3.3 or above, or the Intel compiler icc. In this optimization, the threshold output is computed as src ← min(ones, (t_inc − src)). Here src is a sequence of 8 unsigned bytes of an image to be thresholded. Note that src is overwritten with 0 or 1 values at the end of this operation. ones is an array of 8 unsigned bytes filled with the value 1. t is the threshold value (that is, if a byte in src is <= t, it should be set to 1, and to 0 otherwise) and t_inc be an array of 8 unsigned bytes filled with the value t + 1.

Using the Intel MMX/SSE instructions, the threshold operation can be performed in two vector operations. The subtraction operation (t_inc − src) uses saturating arithmetic, which means that subtraction overflow sets the result to 0, not a negative number. The following example illustrates the thresholding operation, where \( t = 5 \) and src = 1 2 3 4 5 6 7 8:

\[
\begin{array}{cccccccc}
  t_{\text{inc}} & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
 - src & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
  5 & 4 & 3 & 2 & 1 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
  \text{min} & \text{ones} & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
  1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

Similarly, the tessellation operation can also benefit from use of vector operations if the data within a chunk is aligned and laid out in the right way. Hardware acceleration may provide negligible improvements in performance at the level of a single data point or chunk, but when used over very large datasets, their cumulative benefits could be significant. Within OCVM, the mediator service can be configured to layout the image data within a chunk in specific ways through high-level directives such as BGRplanar (where the Blue values of all pixels are stored first, followed by the Green values,
and then the Red values), and **BGRraw** (where all color values of a pixel are stored together before those of the next pixel in the row, and so on). Similar directives can be developed for generic multi-dimensional data chunks. Applications can direct the mediator to store data within input or intermediate data chunks in specific ways if the analysis operations favor specific data layouts.

**Chunk storage:** The final set of configurations of the mediator services within OCVM concern the storage of data chunks within files on local disks. Depending on the characteristics of the disk drives and the local storage system on each node, one can select extreme configurations such as ‘one chunk per file’ (which may result in a very large number of files) or ‘all chunks in one file’ (which may lead to large seek overheads when scanning a single file for each request), or any other intermediate configuration. In our evaluation, we chose an intermediate configuration for the number of chunks to be stored in a file, that was tuned based on the properties of the system. The order in which chunks are stored within files could also potentially impact the seek performance. But since the mediator services maintain the exact location of chunk within a file on disk, we do not configure this parameter and just append new chunks to the end of existing files.

Our experimental evaluation to demonstrate the benefits of the performance optimizations were carried out on a Linux cluster (referred to as XIO) equipped with a mass storage system with total 116.8TB storage. The cluster consists of 16 nodes, each node having two Intel Xeon 2.4GHz processors with hyper threading, resulting in 4 virtual CPUs per node. Each node has 4GB of memory and is connected to 7.3TB storage space – each of the 16 nodes mounts an independent 7.3TB ext3 filesystem. The maximum simultaneous I/O rate is measured to be 4.2GB per second total across
all 16 nodes when all 16 nodes are used simultaneously, and 280MB per second when a single node is used in isolation. However, we found the maximum rate to be less than that for the files we used for our experiments (we achieved a maximum I/O rate of 2.95 GB/sec). We speculate that the cause is greater file fragmentation on the file systems. The nodes were connected using a switched Gigabit Ethernet network.

The goals of our evaluation is to show that our implementation on the XIO cluster, together with performance optimizations provided by customizing the mediator services, can achieve performance close to the application-level maximum raw I/O rate while running the preprocessing operations.

Figure 3.13 illustrates each of these points. The maximum I/O rate achieved on this cluster with our image files was 2.95 GB per second. This rate was achieved by spawning two programs per node in parallel across all 16 nodes which merely read every byte of the same files used in the rest of our experiments on this cluster. The next line down represents our maximum achieved rate with the full prefix sum pipeline; with all of our optimizations, we are able to achieve a 2.68 GB per second input image processing rate, or 91% of the measured maximum I/O rate, on an 8TB input image dataset. This line includes vector operations (known as the SSE instruction set on the Intel architecture) and also uses the Intel icc compiler’s own auto-vectorization feature. The next two lines show the result of running the same pipeline with auto-vectorization but no manual SSE instructions, and the result of running the same pipeline with no vector operations at all. Clearly, the auto-vectorization alone improves performance, but the manual vector optimizations improve performance even

21 We had to rewrite several of the “slower” files using cp just to achieve the 2.95 GB/sec mark; without rewriting, the rate was slower

22 The chunks that make up the 8 Terabytes are stored in a total of 1333 6 GB files across the 16 nodes
Figure 3.13: XIO Cluster Collective End-to-End Preprocessing Rate (16 nodes, 6GB files, local configuration)

...further, thereby justifying the customization of intra-chunk data layout by the mediator service.

### 3.4 Summary and Discussion

This chapter described two approaches to address the first major challenge of data-intensive scientific applications, namely, the large data scales. Both approaches provide a layer of multi-dimensional data services that enable user-defined analysis operations to gracefully scale to very large-scale data sizes by distributing data chunks across nodes of the storage system. In both cases, the services were implemented as extensions to data-centric middleware designed to run parallel dataflow programs (Figure 3.14).

The first set of services explore query scheduling and load balancing strategies to minimize execution time in peer-to-peer storage systems. Our observations here are:
Multi-dimensional indexes and user-defined analysis functions can be stored as objects in the storage system such that any server can process a data analysis query sent by any client.

Query execution by multiple servers can be scheduled (in the absence of explicit control of data storage by applications) to not only balance the load amongst the servers, but also to indirectly influence the data and replica placement policies of the system over time.\(^{23}\)

Dynamic application-driven query scheduling led to better load balance than the static, random and server-driven strategies because it used a better metric to estimate the load on the servers.

The second set of ‘mediator’ services provide scalable I/O virtualization in a cluster environment, and allow explicit application-driven storage and management of data chunks for performance optimizations. The OCVM middleware provided an

\(^{23}\)The Pond prototype system lacked scalability to Gigabyte-scale data. Although our evaluations showed promising results, we could not confirm our hypothesis at large data scales.
instance of existing dataflow systems retrofitted with mediator services configured
for optimal performance of image analysis operations at multi-terabyte scales. Our
observations here are:

• Mediator services helped scale application execution to handle multi-terabyte
  sized images.

• The overheads of slow networks and reliance on virtual memory can be avoided
  by having mediator services coordinating I/O operations on each node. Dy-
  namic repartitioning of chunks makes out-of-core data analysis possible under
  memory constraints.

• Mediator services can be configured to minimize read latency through computation-
  communication overlap. Retrieval, communication and processing of chunks can
  be pipelined so that processes are never blocked on reads.

• Mediator services can be configured to lay data within a chunk on disk in ways
  that are suitable for hardware accelerated parallel processing. When processing
  terabyte-scale image data, the use of vector operations on appropriately orga-
  nized chunk data led to a 10% improvement in application-perceived bandwidth.
CHAPTER 4

APPLICATION CHARACTERISTICS AND HETEROGENEOUS SYSTEM ARCHITECTURES

Data-intensive scientific applications are often executed on shared resources, where the overall resource pool consisting of clusters and collections of servers and workstations is shared by multiple application instances; the appropriate set of resources may need to be provisioned to an application instance prior to execution. These systems are characterized by heterogeneity in their underlying hardware and software architectures. With the levels of scalability provided by the multi-dimensional data services (chapter 3), applications are no longer constrained by the sizes of the datasets they can handle on any given system. However, increased scalability alone may not help applications to meet time constraints on their execution. In such cases, applications that process large data are in need of additional performance optimization opportunities. Understanding the interplay between the characteristics of multi-dimensional data analysis operations and the heterogeneity of resources may reveal application-level performance bottlenecks on specific systems. This chapter presents insights into identifying potential mismatches between the requirements of such applications and the architectural properties of the resources they execute on, when operating at large data scales. Specifically, I show how such mismatches (both at the component and workflow levels) of applications can be resolved to improve performance.
4.1 Workflow Scheduling

Application scheduling involves mapping of the data analysis tasks in the application workflow onto one or more compute resources in the system in order to minimize end-to-end execution time for an application. Workflow scheduling strategies must make optimal utilization of the available resources so as to meet critical time constraints put forth by the scientists. Traditionally, application scheduling on clusters and supercomputer centers has been achieved through the use of batch schedulers like Condor [105] and PBS (see process-centric middleware in figure 1.1) which view application instances as a bag of independent tasks. The strategies for scheduling such bag-of-tasks applications are typically based on the estimated computation time of the jobs and the system state (e.g., queue wait times). Workflow applications add an extra dimension to the scheduling because of the dependencies between the tasks as dictated by their DAG structure. Workflow scheduling, sometimes referred to as meta-scheduling, is carried out hierarchically where each component or coarse-grain analysis task in the workflow is first mapped to a specific set of resources in the system. For e.g., Grid workflow middleware such as Pegasus and DAGMan map each component of an application workflow instance onto an appropriate site on the Grid. Thereafter, local job schedulers at each site are responsible for mapping and scheduling the tasks mapped to their respective sites. Given that different sets of resources could be characterized by differences in their system architecture, workflow scheduling strategies need to take this heterogeneity into account when mapping components to resources. A large body of research exists in the area of job scheduling for heterogeneous environments and several job scheduling strategies have been proposed
for mapping/scheduling application workflows onto heterogeneous systems. However, these are mainly in the context of computationally-intensive workflows that are characterized by hundreds of thousands of light-weight interdependent tasks. These strategies primarily seek to minimize end-to-end execution time of the applications; other goals may include optimizing multiple criteria (e.g., makespan and throughput) for the applications in the presence of multiple constraints (e.g., on latency). There are three potential drawbacks of using these strategies in the context of data-intensive application workflows such as the ones we encounter in our target application domain:

(1) These strategies are generally application-agnostic and assume an application model which treats all analysis components equally from a functional point of view. That is, the model may distinguish between analysis processes based on estimated execution times and amount of data output produced, but they do not differentiate between components from a functional point of view. However, we observe that the specific characteristics of different analysis operations in an application workflow can be used to improve overall execution time. As an example, we continue our evaluation of analysis operations borrowed from the PIQ workflow (use-case scenario 1 in section 2.2.1) using the ‘local’ and ‘remote’ execution configurations as described in section 3.3.3. Now, we can select either local or remote execution strategy on a per-operation basis. Our goal is to determine which mapping combination (of operations to the RII-MEMORY and RII-COMPUTE cluster nodes) results in minimum end-to-end execution time. We first determined the best placement for each individual component in the workflow. Regardless of their placement, the output from each component is staged to disk on the RII-MEMORY cluster prior to executing the
next component in the workflow. As seen in Figure 4.1, the zproject and stitch components execute faster in the local configuration, while the normalize and autoalign components were faster when executed on the same number of nodes in the remote configuration. Having components observe performance improvements in the remote configuration is interesting because the measured time in this configuration also includes the time to read and write data from remote disk over the network. So the question arises as to what is it about the normalize and autoalign components that makes them run faster on the RII-COMPUTE nodes? To understand this better, we take a closer look at the analysis requirements of these components and the properties of the resources. The dedicated 10 Gbps connection between the two clusters means that the overhead of reading and writing data in the remote configuration may not

![Data Analysis Component](image)

Figure 4.1: Overall time taken: Local v/s Remote component placement for PIQ transformation tasks (5 GB image, 16 nodes overall)

24 A similar trend was observed when we varied the number of nodes used to execute components on each cluster, and also well when the data size increased [68].
be significant for Gigabyte-scale datasets. The nodes on the RII-COMPUTE cluster are equipped with faster (3.2 GHz Intel Xeon) processors and high-speed interconnect (switched Infiniband) as compared to the nodes on the RII-MEMORY cluster used in our evaluation. The architectural properties of the RII-COMPUTE cluster nodes and the high-bandwidth dedicated link between the clusters suggest that components with relatively large amounts of computation and communication between processes are likely to perform better on these nodes. Profiling of the individual analysis compo-

![Image](image.png)

Figure 4.2: Percentage of time spent in computation by the PIQ transformation tasks (5 GB image, 16 nodes overall)

ments was performed, and the results (shown in figure 4.2) revealed that the normalize and autoalign components spend a majority (> 70%) of their overall execution time purely on computation. Also, there are frequent data exchanges between multiple filter instances for these components. An exception to this observation is the warp component which also spends a majority of its time doing computation, yet is faster in the local configuration. This is because the warp component is also characterized by

109
the need to frequently access disk for much larger amounts of data during execution as compared to normalize and autoalign. Therefore, if an application component’s

![Workflow mapping with optimal configuration for individual components](image)

Figure 4.3: Workflow mapping with optimal configuration for individual components (256 GB image, 32 nodes overall)

execution is characterized by large amounts of computation and communication relative to disk I/O, and if a set of resources in a heterogeneous system are equipped with relatively faster processors and interconnects, then, mapping such components to these resources can lead to performance improvements. We can use this knowledge about the analysis operations to adopt a hybrid ‘local and remote’ component placement scheme, where each component is mapped to resources depending on its optimal configuration, then the results in figure 4.3 show that the overall execution time of the workflow decreases.

(2) The strategies **do not factor-in the data movement** between components. While the workflow scheduling strategy used in the above example yields performance benefits by matching analysis requirements with the system architecture properties,
the evaluation does not consider the overheads of data transfer between analysis operations incurred while executing at different cluster sites. For data analysis applications characterized by large amounts of intermediate data exchanged between components, minimizing the execution time of each individual component alone may not always add up to optimal end-to-end execution times for the application workflow. In the above example, assume that the execution of the normalize operation under the remote configuration brought about an improvement in its execution time by $t_c$ seconds over its execution under the local configuration. Now, assume that the overhead of transferring input data to the normalize component from one cluster to the other over the network and that of transferring its output data back to the first cluster is $t_d$ seconds. If $t_d > t_c$, then the performance improvements brought about by placing the normalize component on a remote set of resources are negated by the data transfer overheads.

(3) The strategies are mostly static and require the existence of accurate analytical performance models to estimate the execution times and other metrics associated with the application workflow based only on metadata about the input datasets. In large-scale data-driven applications such as the ones in our target domain, the execution time also depends on the nature of the computation performed by the analysis tasks and the content of the input data. In data-driven applications, the data drives the analysis tasks within the workflow. So, in many cases, the actual number and nature of tasks corresponding to different components in the workflow may depend on the input data content, and may not be known a priori. In general, it is not feasible to assume the availability of accurate performance models for data-intensive applications on all kinds of heterogeneous system architectures, much moreso
for our target multi-dimensional data analysis applications where individual analysis
task codes may be developed by different researchers.

Given the significance of the data transfers towards the end-to-end execution time,
the transfer of data between workflow components needs to be made an integral part
of the scheduling strategy for such applications. Data-intensive workflow scheduling
strategies need to be cognizant of the amounts of data exchanged between components
and must dynamically determine component placement and mode of data transfer
depending on the estimated transfer costs. In contrast to compute-intensive work-
flow scheduling, scheduling of data-intensive application workflows onto resources in
heterogeneous computing environments is a relatively less explored area of research.
This latter form of scheduling to minimize execution time, with or without added con-
straints is considered to be NP-complete, and most proposed strategies are heuristics
that focus on minimizing extraneous data transfers among clusters and Grid com-
puting sites. Given the circumstances, the heuristics look to exploit different forms
of added knowledge to obtain as accurate an estimate of the component performance
and data exchanges as possible, using the input metadata. Some heuristics rely on
the past history of execution to guide future estimates, but such a strategy requires
homogeneity of application execution context over time. Other heuristics rely on the
assumption that the use of model learning and relative fitness to measure perfor-
mance of applications running on different systems is accurate enough. The end goal
of these heuristics is to obtain a workflow scheduling strategy that is believed to be
sufficiently accurate over a wide range of inputs. In my opinion, given the complex
nature of multi-dimensional data analysis applications, and the ad hoc requirements
of the different components that make up the workflows, the widespread application of such workflow scheduling strategies may not contribute to consistent improvements in performance. A user-directed mapping of components to resources, where users can apply their knowledge of the task characteristics and system properties to offer hints towards high-level component placement, is likely to yield reasonably good schedules as compared to the heuristic-driven (or system-directed) strategy. Figure 4.4 shows a user-directed optimal placement of components in the PIQ workflow that also takes into account data transfer between sites. Here, the components shown in blue execute on RII-MEMORY, while those in red are placed on RII-CLUSTER. The arrows indicate the data dependencies between pairs of components, and the numbers beside the components and links show the sequence of execution in the workflow. For e.g., the 2,3,4 along the link from zproject to normalize means that the data transfer between

Figure 4.4: User-directed component placement that also considers data transfer
these two components running on different sites occurs at the same time as events marked 2 (data transfer between zproject and prenormalize within RII-MEMORY), 3 (the prenormalize component), and 4 (data transfer between prenormalize and normalize). In this way, this placement of components avoids unnecessary data transfer overheads and where required, tries to overlap computation and communication to the extent possible.

**Summary.** Data-intensive workflow scheduling is required to minimize application execution time and to meet time constraints on the execution when scalability alone is insufficient. The scheduling strategies aim to map individual analysis components of the workflow onto resources of the underlying system in a manner that minimizes end-to-end execution time. Regardless of whether the component placement is user-directed or system-directed, the strategy takes into account both the estimated execution time of components on different heterogeneous architectures, as well as the data transfer overheads between components under different placement configurations. An important side-effect of this mapping is that components may now be constrained by the set of resources they are bound to for execution. The estimates of execution time and data transfer are based solely on the metadata properties of the input data to the components and the role of the components in the end-to-end workflow context, and do not consider the data content or the functional aspects of the analysis operation (i.e., the nature of the processing operations performed on the data). As such strategies are likely to sacrifice local (component-level) optima in the interests of obtaining a global (workflow-level) optima, components are likely to be assigned to resources with architecture configurations that are not compatible with their ‘style’ of analysis. Given the heterogeneity in hardware and software
architecture among the resources, it is highly possible that there are one or more mismatches between the requirements of the analysis tasks and the properties of the resources assigned to it.

In this chapter, I investigate the execution characteristics of a set of complex analysis operations commonly performed on large-scale multi-dimensional data. I then identify potential mismatches between the requirements for efficient analysis and the underlying system that go against the performance benefits suggested by the workflow scheduling strategy, and prevent the applications from meeting time constraints. Mechanisms to resolve such mismatches through application-level configuration on different systems are presented.

4.2 Application-level Configurations

In this section, we use example instances borrowed from our motivating applications where the workflow scheduling strategy employed potentially constrains one or more analysis components (or ensembles of components) to execute on specific sets of resources. Incompatibility between the analysis requirements and the underlying system properties prevent these application instances from meeting time constraints on their execution. The analysis requirements could be associated with components or sub-workflows or entire applications, while the system properties could refer to its hardware and/or software architecture.

4.2.1 Computation of Global Data Structures

Scientific multi-dimensional data analysis applications often require preprocessing or preparatory data analytics performed on entire datasets in order to compute a global data structure such as an $n$-dimensional prefix sum index or a histogram
over the data. These structures are then used to speed up the execution of subsequent operations such as querying and online monitoring of the large datasets. The computation of such global data structures may be characterized by certain specific requirements of these preprocessing analysis operations. Dependencies may exist between different regions of the multi-dimensional data space. That is, the global data structure value of a region $R_0$ in the dataset may depend on the values of several other regions \{$R_1, R_2, \ldots, R_n$\}, and may not be computed until the values for these regions are computed first. Once all dependencies for data region $R_0$ are resolved, its global data value is then computed and potentially used by the application for computing values of other regions which depend on it. We illustrate a potential mismatch between such analysis operations and the system properties in the context of the 2D prefix sum generation operation borrowed from the PIQ workflow. The last of the preprocessing tasks performed ahead of querying in the PIQ workflow is the generation of 2D prefix sums for the tessellated image data. The 2D prefix sum at each point $(x,y)$ of an image is given by:

$$P_{Sum}(x, y) = \sum_{i=1}^{x} \sum_{j=1}^{y} value(i, j)$$

These prefix sums are stored and subsequently used for efficient execution of query operations [107].

**Global (Full) Prefix sum:** The global 2D prefix sum values across all pixel points in the image data are computed by adding the values serially along each dimension, i.e., (1) A 1D prefix sum from left to right along each row of the image followed by (2) A 1D prefix sum from top to bottom along each column of the image. When data chunking is used at large data scales, the chunks are distributed across multiple nodes in the system. On a **shared-nothing system architecture**, one of more application
filter instances on each node are responsible for computing the full prefix sum values for pixel points within chunks stored on their respective nodes. When computing the full prefix sums, there exists an inherent dependency of each chunk on the chunks to the left of and above it in the image. Thus, a filter handling chunk \((x, y)\), where \(x > 0\) and \(y > 0\), has to wait for values from the filter(s) handling chunks \((x - 1, y)\) and \((x, y - 1)\). So this analysis characteristic introduces a diagonal wave of dependencies between chunks that propagates from left to right along each row and from top to bottom along each column. As a result, the filter instances operating on some of the chunks are stalled waiting for their dependencies to get resolved. The stacked configuration is preferred for distribution of data chunks to minimize communication overheads between the nodes. But for computing the global prefix sums, exchange of data between filter instances on different nodes of the system is inevitable. The bigger problem is that the dependencies may lead to under-utilization of the nodes by the stalled filter instances [97]. Thus, the dependency characteristic of the full prefix sum and the shared-nothing architecture of the system that forces communication over the network are mismatches that make this operation an expensive one at large data scales.

**Local (Partial) Prefix Sum**: To address the above mismatch, we proposed an approach where the prefix sum filter instances compute only local 2D prefix sums within each chunk, as opposed to global prefix sum across the entire dataset. For every point within a chunk, each filter instance computes and stores only the 2D prefix sum values local to that chunk. Mismatches are resolved through dependency relaxation and communication elimination. This technique overcomes the stalls that arise from the dependencies present in full 2D prefix sum computation. Since each
filter can compute the local prefix sums for a chunk independent of other chunks, there is no dependency between the filters. This technique helps to eliminate the idle time that can affect the performance of the full prefix sum technique. Also, since all filter computations are performed only on local data on each node, this approach is practically communication-free. Figure 4.5 shows the improvements in execution time brought about by the dependency relaxation in the local prefix sum. Thus, by having the prefix sum generation step produce only partial results (that is, values local to each chunk), significant performance improvements can be observed at large data scales. This improvement in prefix sum generation time by resolving the mismatch comes at the expense of increased complexity of the querying stage that follows.

**Query Execution:** A query in our system is broken down into a sequence of points that make up the region of interest within an image. Query execution involves the retrieval and computation of the full prefix sum values corresponding to these query points in the input (tessellated) image. When the full prefix sums are generated, a

---

25Decreasing tessellation size along X-axis is equivalent to increasing the input data to the prefix sum step.
query filter performs lookup operations to determine the values at each query point. If local prefix sums are generated, then, the full prefix sum value at each query point must now be computed at runtime, by combining relevant local prefix sum values. Consider a single query point \((x, y)\) in the tessellated image. Let \((x, y)\) lie within chunk \((a, b)\) of the tessellated image as shown in Figure 4.6. Let \(PSum(x, y)\) denote the full prefix sum at \((x, y)\) and \(LPSum(x, y)\) denote the local prefix sum at \((x, y)\) within chunk \((a, b)\). The \(LPSum\) value at the lower right corner point of any chunk \((i, j)\) will be equal to the sum of all the values within that chunk. These points are denoted as \(lrc(i, j)\) and are highlighted in the figure using darkened boxes. \(lrc(i, j).x\) and \(lrc(i, j).y\) respectively denote the horizontal and vertical coordinates of \(lrc(i, j)\) in the image. In other words, \(lrc(i, j).x\) gives us the value of the horizontal coordinate for the rightmost column of chunk \((i, j)\) and \(lrc(i, j).y\) gives us the vertical coordinate.

Figure 4.6: Computation of full prefix sum using local prefix sums (with 16 total chunks)
value for the bottom row of \((i, j)\). Our proposed algorithm used to compute the full prefix sum at a given query point is shown in Figure 4.7. The full prefix sum at

<table>
<thead>
<tr>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query point ((x, y)) in image.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Find ((a, b)): the chunk within which ((x, y)) lies</td>
</tr>
<tr>
<td>2 (PSum(x, y) \leftarrow 0)</td>
</tr>
<tr>
<td>3 if (a &gt; 0) and (b &gt; 0)</td>
</tr>
<tr>
<td>3.1 (PSum(x, y) \leftarrow PSum(x, y) + \sum_{i=0}^{a-1} \sum_{j=0}^{b-1} LPSum(lrc(i, j).x, lrc(i, j).y))</td>
</tr>
<tr>
<td>4 for each chunk ((i, j)) in {((a, 0), \ldots, (a, b - 1)})</td>
</tr>
<tr>
<td>4.1 (PSum(x, y) \leftarrow PSum(x, y) + LPSum(x, lrc(i, j).y))</td>
</tr>
<tr>
<td>5 endfor</td>
</tr>
<tr>
<td>6 for each chunk ((i, j)) in {((0, b), \ldots, (a - 1, b)})</td>
</tr>
<tr>
<td>7.1 (PSum(x, y) \leftarrow PSum(x, y) + LPSum(lrc(i, j).x, y))</td>
</tr>
<tr>
<td>7 endfor</td>
</tr>
<tr>
<td>8 (PSum(x, y) \leftarrow PSum(x, y) + LPSum(x, y))</td>
</tr>
</tbody>
</table>

Figure 4.7: Algorithm to compute full prefix sum at a point using local prefix sums a point is computed as the aggregate of the local prefix sums within four disjoint regions, numbered from R1 through R4 in Figure 4.6 [97]. The query filter can then use the full prefix sum values to determine the result of the query [107].

We measure and compare the cost for issuing queries against local and full prefix sums to see if the overheads of our proposed querying algorithm are tolerable to the end-user. Table 4.1 shows the results of performing a series of queries against the 2x2 tessellation prefix sum from Figure 4.5. We ran 20 queries using a random set of 4, 128 and 1024 query points within the image (the same random points were used for both local and full prefix sum tests to make it a fair comparison) and averaged
the runs together to produce the timings. Observe that the full prefix sum strategy outperforms the local prefix sum strategy in all tests. This is due to a combination of factors, including fewer chunk files to open and fewer individual seeks to make to read individual data points. Given that the local prefix sum can outperform the full prefix sum during prefix sum generation, and the full prefix sum can outperform the local prefix sum during querying, the question remains as to which strategy to use. With this example (2x2 tessellation), the difference in run times for the preprocessing phase was 120 seconds. For the 4 query point case, the user would need to make more than 1500 queries for the full prefix sum to emerge as the better overall solution ($1500 \times (0.23 - 0.15) = 120$). For the 1024 query point case, the user would need to make around 39 queries. Therefore, only if the number of queries is large would the user realize a noticeable improvement with the full prefix sum. Therefore, the local prefix sums introduce negligible overheads to querying unless the number of queries are going to be very large by interactive standards.

**Conclusions.** An important class of data analysis operations in our target domain are those that compute global data structures over multi-dimensional datasets. This computation may need to be performed every time the user modifies criteria for individual dimensions. Hence, the computation must be carried out efficiently to

<table>
<thead>
<tr>
<th>#query</th>
<th>full prefix sum</th>
<th>local prefix sum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>opens</td>
</tr>
<tr>
<td>4</td>
<td>0.15 s</td>
<td>2</td>
</tr>
<tr>
<td>128</td>
<td>0.30 s</td>
<td>104</td>
</tr>
<tr>
<td>1024</td>
<td>0.56 s</td>
<td>547</td>
</tr>
</tbody>
</table>

Table 4.1: Query Performance results
meet time constraints, especially at large data scales. When this computation is accomplished by many processes executing in parallel across nodes of a shared-nothing architecture system, then the dependency between processes may cause processes to stall waiting on data from other processes. To resolve this mismatch, our proposed application-level configuration is to have the analysis operation compute only partial data structures across regions of the dataset instead of computing the global data structure across the entire dataset. Dependency relaxation and communication elimination can be used to improve performance of such preprocessing analysis operations. Downstream operations like querying can be configured through application-level services with relevant optimizations, so that they do not witness any noticeable overheads in execution time.

4.2.2 Mapping Multi-dimensional Datasets

A majority of scientific data analysis workflow applications involve the mapping of spatial regions of a source multi-dimensional dataset onto spatial regions of one or more target (or template) multi-dimensional datasets. The main goal of these mappings is to establish the qualitative equivalence of studies conducted on multi-dimensional datasets obtained from different sources and under different experimental conditions. These mappings are achieved through the application of transformation functions on data entities within the datasets involved. These functions can be classified in several ways that are characterized by differences in their computational complexity: (i) The same function could be used to map all points within the dataset. Alternatively, different mapping functions could be used for each point. An intermediate scenario is one where different functions are used to map different spatial
regions, but within a given region, the same function is used to map all the points,
(ii) The functions could range from simple affine transformations that do not modify
the structure of the data entity being mapped, to shear transformations that modify
the structure along specific dimensions of the data, to complex polynomial transfor-
mations that are non-rigid and do not preserve the structure of the data space being
mapped, (iii) The functions could be classified as forward or inverse, depending on
whether the transformations are applied on the input or output data spaces.

Such analysis operations are typically characterized by their two-phase compu-
tation – first, a ‘map’ phase applies the transformation function on different data
entities (point or regions) of the source dataset and stores the mapping information
in some mapping data structure (MDS), a second ‘copy’ phase then uses this mapping
information to copy the values from data entities in the source onto entities in the
target datasets. In the map phase, if the transformation functions are simple (e.g.,
the same function applied to all data points or regions of the dataset), then the in-
termediate mapping data structure need not necessarily be maintained. Instead, the
mapping information could easily be derived on-demand. If the transformation func-
tions depend only on the positions of the data entities within the multi-dimensional
coordinate system, then the mapping information can be obtained using only the
structural metadata about the datasets involved. That is, we can compute the MDS
for the dataset without actually reading source datasets off disk. In fact, frameworks
like the Active Data Repository (ADR) use mapping information to optimize
storage, retrieval and processing of data when mapping large multi-dimensional data

\[ \text{In certain cases, additional analysis operations may be performed on the source values before}
\]
\[ \text{copying them to their destination locations.} \]
spaces. The mapping information itself will vary depending on whether we use forward or inverse transformation functions. Forward mapping data structure entries can maintain the value associated with the source data entity and the destination locations. Inverse mapping data structure entries maintain mapping between source locations and destination locations and between source values and destination locations.

In more complex mapping operations, the transformation functions depend not only on the structural metadata of the datasets but also on auxiliary user input in the form of transformation control points or landmarks which are nothing but user-specified explicit transformations for a small set of points in the datasets. That is, the transformation function for each data point or region within the dataset is derived only at runtime based on its spatial proximity to the set of control points. In such cases, the mapping information for the entire dataset cannot be computed beforehand without the user input. The scheduling and data movement optimizations provided by frameworks like ADR [71] or MapReduce may not work for such mappings. The MDS for such complex transformations may vary in size depending upon the granularity at which the transformations are applied. At large data scales, the manner in which the system stores and utilizes the information within the MDS is crucial to the performance of the operation. We illustrate potential mismatches that may arise between such complex mapping operation requirements and the system’s hardware architecture in the context of the 2D inverse image warping operation borrowed from the PIQ workflow.
**Image warping** or image registration is a computationally-intensive image processing operation and is part of multiple application use-case scenarios described in chapter 2. Warping geometrically transforms image data, typically, to conform to a standard pre-defined canonical atlas, such as the Allen Reference Atlas—a high-resolution anatomical 3D atlas of the mouse brain, to facilitate easier comparison between multiple images of the same subject type. Warping involves the application of a transformation function between points in the input and output images. In 2D, mapping functions $f_1$ and $f_2$ map a point $\langle x, y \rangle$ in one image onto a point $\langle u, v \rangle$ in the other image as follows:

$$[u, v] = [f_1(x, y), f_2(x, y)] \quad (4.2)$$

In some warping techniques, a forward mapping function is defined and applied to each input pixel to produce its output pixel location. In other techniques, an inverse mapping function is defined and applied to each output pixel location to determine which input pixel to draw from to produce the output pixel. Inverse mapping prevents "holes" from occurring in the warped image. We focus on the inverse 2D warp, i.e., for image data from a single focal plane at a time. The 2D warping of each plane is independent of the other planes. In the PIQ workflow, inverse warping is used and is based on a set of control points provided by the user. For each control point, the user identifies its position in the input image, called "start" and its corresponding position in the output image, called "end". The start and end of each control point indicates the user’s requested change in the image. These control points as a set describe how the image should morph. Figure 4.8 shows how the control points can move pixels at or near the start in the original image should move to a corresponding position at or near the end in the warped image.
be specified to warp an image of a portion of the brain so that it fits a standard brain atlas. On the left we see the unwarped image (colored region) overlay the atlas. On the right, we see the same image warped so as to fit the atlas dimensions.

Figure 4.8: Warping of a brain mosaic to a standard brain atlas

The inverse warping algorithm works as follows: given a set of control points, approximate inverse mapping functions are generated that characterize the spatial correspondence and map the remaining points in the image. Given any pixel $p_o$’s location in the output space, the function helps determine which candidate input pixel $p_i$ will contribute to pixel $p_o$ with minimum error (the map phase) and store this mapping information. The RGB color values at $p_i$ are then assigned to $p_o$ in the copy phase. The warping algorithm must iterate over the output space and determine which areas of the input space to read from. This correspondence is modeled using low-order polynomials. The Weighted Least-Squares with Orthogonal Polynomials technique [111] is used to determine the inverse mapping. The greater the number of control
points, the more complex the set of polynomials, and hence, greater the computation time.

**Implementation:** Our parallel implementation of the inverse warping algorithm for chunked image datasets on a **cluster system with a shared-nothing architecture** is accomplished through one or more warping filter instances per node that compute the mapping information for each chunk that the node owns. Each entry within this MDS maps a point $p_o$ in the output image to a pixel point $p_i$ in the input image and takes the form of a vector pair: $\{p_o(x_o, y_o, z_o) \leftarrow p_i(x_i, y_i, z_i)\}$. Each such entry is communicated as a request to a filter instance on a node that owns the chunk containing pixel point $p_i$. Mapping information for all output chunks are computed. At this point, the MDS is complete and its entries are shuffled amongst the filter instances as per the communication policy specified. Now, each filter instance extracts the pixel color values corresponding to every entry request it received by reading the corresponding input image chunk. The extracted values are then maintained in a second MDS, where each entry takes the form of a vector pair: $\{p_o(x_o, y_o) \leftarrow p_i(R, G, B)\}$. Each such entry is communicated as a response to the filter instance that sent the request (i.e., a filter instance on the node that owns the chunk containing $p_o$). Finally, after all requested values are extracted and communicated, the second MDS is also complete. Each filter instance then assigns the color value to the output point $p_o$ for every entry it receives. Even though data chunks are not communicated between nodes, the parallel warping algorithm operating on out-of-core image chunks is communication-intensive, with potentially large numbers of entries being exchanged among the filter instances. Reading input chunks on a per-point request basis may not be feasible and could potentially lead to thrashing if neighboring points are mapped.
to different input chunks. To minimize the execution time of inverse image warping, our goals are to minimize the number of times an input chunk is read in response to a request and to minimize the number of times each output chunk is written to disk. We rely on the mediator services described in chapter for system-level data chunk management.

**Configurations and Mismatches:** There are different application-level configurations that can be adopted for the parallel inverse image warping algorithm. These configurations differ in terms of how the intermediate mapping data structure is handled during execution. Since the mapping are maintained on a per pixel point basis, the size of the MDS could be very large when warping images with billions of pixels, potentially larger than the input image data itself. Hence, in our first proposed configuration, known as the **serializable mappings** (SM) configuration, we maintain the billions of MDS entries in the form of serializable log files on the local disks of the cluster nodes. That is, when filter instances receive requests containing mapping entries, each filter will store them sequentially in a local disk log file, using one log file per input chunk it owns. Once the MDS is completely built, each filter will iterate over its log files one at a time, read the input chunk corresponding to each log file and satisfy all pixel requests in the log file thereby reading every input chunk only once to service all requests. In doing so, the filter communicates each RGB pixel value extracted from the input chunk to the compute node that made the request. When all MDS entries within a log file have been iterated over, and the pixel requests serviced, the log file is purged. Upon receiving an RGB pixel value targeting an output pixel location, each filter instance stores the pixel’s color values

---

28We do not know upfront which input chunks will be needed to serve requests, as the transformation functions are determined only at runtime.
sequentially in another local disk log file ($logfile_b$), again organized with one log file per output chunk. Finally, when all pixel values have been communicated and all instances of $logfile_b$ (the second MDS) are completely built, each filter will, one at a time, replay its $logfile_b$ files, finalize the corresponding output chunks, write it out to disk (exactly once per output chunk), and purge the log file.

At large data scales, mismatches exist between the analysis requirements of the SM configuration and the hardware and software architecture of the systems. The SM configuration transfers very large amounts of mapping data to and from local disks. Each pixel may be represented by 3 bytes, one per color value. On the other hand, each mapping entry within the $logfile_a$ MDS is 40 bytes, and each mapping entry within the $logfile_b$ MDS is 19 bytes. For a terabyte-sized image, this configuration would require up to a maximum of 13 terabytes of additional storage for the MDS entries during execution. Although disks are cheap, additional storage requirements of the order of several terabytes may not be met by systems with limited storage capacity. If the execution is being carried out on a system with low bandwidth access to local disks, then the overheads of managing MDS data on disk in this configuration could potentially be greater than the actual time spent in useful computation. Even when the system presents no issues (i.e., it has high disk bandwidth, cheap storage), the software architecture on the system may not scale to large MDS entries. The mediator services provide scalability only for multi-dimensional data chunks. However, the MDS entries are serialized log files, which are not multi-dimensional, and involve a large number of small reads/writes to the log files. Unless the middleware on the system includes a specialized set of services (such as [14]) to optimally manage
structured, serial data, performance of the warping operation could suffer because of the application’s mismatch with the software architecture.

To resolve these mismatches, we propose a second configuration, known as the on-demand mapper (ODM) configuration, where we seek to maintain MDS entries corresponding to a small set of ‘active’ output chunks in memory at a time, and serve their input chunk requests on-demand. The formal algorithm is shown in figure 4.9. Here, each filter instance initially computes pixel mappings for a set of $N$ output chunks that it owns. During this step, it also computes the set of all input chunks required to create the pixel values for only these active output chunks. If the input image is partitioned regularly into rectangular 2D chunks, then the complexity of finding the input chunk for a given input pixel is $O(1)$. Each needed input chunk is retrieved into memory, one at a time. All needed pixels from each input chunk are applied to whichever $N$ output chunks need them, before the input chunk is evicted from memory. Once all the input chunks have been retrieved and processed, the output chunks have been created and can be written to disk. The filter then moves on to the next set of $N$ output chunks it owns. Since the MDS entries are always maintained in memory and never stored to disk, ODM does not suffer from the potential mismatches that the SM configuration did.

The ODM configuration reduces the number of times an input chunk must be read. If there are $T_n$ output chunks, then in the worst case, each input chunk must be read $\frac{T_n}{N}$ times. In addition, within a “run” of handling $N$ output chunks, this algorithm is in effect a perfect scheduler; it schedules all accesses for a given input chunk such that they all fall sequentially. A potential mismatch of ODM’s analysis

---

29 the value of $N$ is set based on the size of the chunk, and on the amount of physical memory available on a node.
Input:
Input image $I$, with corresponding array of chunks $T_I$
Number of $x, y$ chunks as $x_{max}, y_{max}$
Number of output chunks $N$ to process together

Algorithm:
01 From $I$, implicitly derive output image $O$, with corresponding set of chunks $T_O$.
02 $M \leftarrow \{\}$ (dictionary mapping input chunks to an array of [input pixel location, output pointer] tuples)
03 $A \leftarrow \[]$ (array of active output chunks)
04 $n \leftarrow 0$ (size of $A$)
05 for $y = 0$ to $y_{max} - 1$
06 for $x = 0$ to $x_{max} - 1$
07 $t_I \leftarrow T_I(x, y)$
08 if on_local_disk($t_I$)
09 $t_O \leftarrow \text{“allocate new output chunk”}$
10 append $t_O$ to $A$
11 for every pixel $p_O$ in $t_O$
12 compute input pixel $p_I$ that maps to $p_O$
13 $ptr \leftarrow p_O$’s array location within $t_O$
14 append mapping $t_I \rightarrow [p_I, ptr]$ to $M$
15 $n \leftarrow n + 1$
16 if $n = N$ or ($x = x_{max} - 1$ and $y = y_{max} - 1$)
17 for every $t_I$ in $M$
18 read $t_I$ into memory from local/remote disk
19 for every $[p_I, ptr]$ associated with $t_I$
20 set $ptr$ to RGB value at $p_I$
21 for every $t_O$ in $A$
22 write $t_O$ to local disk
23 deallocate $t_O$
24 $M \leftarrow \{\}$, $A \leftarrow \[]$, $n \leftarrow 0$

Figure 4.9: ODM configuration for parallel 2D inverse image warping
requirements and the system properties may arise as the memory requirements of this configuration is higher than that of SM. If $N$ defines the number of output chunks to be processed at a time and $S$ is the size in bytes of an average chunk, then the total amount of main memory required during processing is equal to the sum of (1) $N \times S$ for the output chunks, (2) $N \times S \times \frac{32}{3}$ for the stored mappings and (3) $S$ for the one input chunk being read at a time. If the workflow scheduling strategy constrains the warping component to execute on a set of compute nodes equipped with low amounts of physical memory, then this mismatch could prevent the ODM configuration from being used. Figure 4.10 shows the performance implications of

![Figure 4.10: Effects of large chunk size leading to increased memory requirements of ODM](image)

increasing the size of each chunk, while keeping the number of chunks in X and Y fixed at 4, and keeping the number of compute nodes fixed at 4, thereby increasing the memory requirements of ODM. Results obtained on the RII-MEMORY cluster show
that the ODM configuration outperforms SM for most of the sizes; however, with a chunk size of 768MB, ODM ran out of memory (hence the missing data point).

Our final configuration, known as the **Asymmetric Traveling Salesman Problem scheduler** (ATSP) configuration seeks to avoid the potential mismatches of the earlier SM and ODM configurations. ATSP does not maintain any pixel-level MDS entries during execution. However, ATSP makes two passes over the output dataset and does the mapping phase twice, thus performing two rounds of computation of the transformation functions. In the first pass, each filter *implicitly* partitions each output chunk it owns, into as few disjoint subchunks as possible (where each subchunk contains one or more neighboring points in row-major order in the chunk), such that the set of all input chunks required to map all points in each subchunk can fit in memory on that node – we denote this set of input chunks corresponding to a subchunk as the “input-set” of that subchunk. In most cases, there exists a fair amount of overlap between the input-sets of different subchunks within an output chunk. Each node maintains an in-memory lookup table that maps each subchunk to its input-set. Next, each node determines a local schedule of data movement that minimizes the number of input chunk reads (An input chunk must be read once for every input-set that contains it). Since the input-sets overlap each other, each input chunk can be cached and reused for servicing multiple subchunks. To minimize this data retrieval overhead, each node traverses the subchunks in its output space in such an order that results in high overlap of the input-sets between successive subchunks. This problem translates to an instance of the Traveling Salesman Problem (TSP) [73], where, for a node, each subchunk is a vertex of a graph and the set difference between
the input-sets of any two subchunks is the weight of the edge between the corresponding vertices in the graph. Hence, the cost of traveling from one vertex to another is the amount of input chunk data that needs to be read to service all requests in the destination’s input-set. Since set difference is not always commutative, the resultant graph is asymmetric (hence the name ATSP). Each compute node then determines a least cost path that starts at any subchunk and traverses every subchunk exactly once. We used the effective LKH [52] implementation of the Lin-Kernighan heuristic [76] to solve this TSP instance, as explained in [69]. TSP-solvers were used to solve the ATSP instance [69]. Once the least cost path for output space traversal is determined, each node makes its second pass over the output chunks it owns; only, this time it reads input chunks in the order determined by the ATSP scheduler. For a subchunk within an output chunk, we read its entire input-set at once (by definition, this is guaranteed to fit in memory). We re-compute the input pixel for each point within the subchunk and assign the color values to the output point.

Figures 4.11(a) and 4.11(b) show the relative performance of the warp component when configured to run under the three different configurations described above. These experiments were conducted on the RII-MEMORY cluster. First of all, our results show that each configuration for the warp component can scale linearly with both the number of compute nodes used and the image sizes. The results also show that for the RII-MEMORY resource, the ODM configuration outperforms both SM and ATSP. Hence, for best performance on RII-MEMORY, the warp component should be configured using ODM. This is because the RII-MEMORY nodes have large amount

Actually, the problem is one of finding the shortest Hamiltonian path in an asymmetric graph, but this can easily be reduced to an instance of ATSP.
Figure 4.11: Warp component performance under three different configurations of memory (8 GB), and this suits a configuration like ODM which stores all active output chunk mappings in RAM on the node. The SM configuration tries to create a global schedule of data movement that eliminates all communication of whole chunks between compute nodes. Instead, only the mapping requests and values are communicated. However, the storage overhead of this algorithm is high, and it becomes infeasible to handle large images and large log files when storage is a constrained resource. Moreover, larger the log files, the file handling overhead (logging the mappings and replaying the logs) assumes significance. The ODM configuration reduces the number of times an input chunk must be read. However, it has a high memory footprint as all mappings for a set of $N$ output chunks need to be stored in memory at a time. This is useful when we have large amounts of RAM, but low disk bandwidth (as was the case for RII-MEMORY). The ATSP variant does not store any mapping entries on disk, and even the in-memory dictionary mapping subchunks to input-sets is small in size. However, the mappings are computed twice – this may be detrimental to the performance when we have slow processors.
Conclusions. An important class of data analysis operations in our target domain are those that compute mappings between multi-dimensional datasets. When the mapping is directed by user-specified control points, a defining characteristic of these operations is the creation and maintenance of an intermediate Mapping Data Structure (MDS). Depending on the granularity of the transformation function used for the mapping, the MDS could grow much larger than the input dataset itself. The choice of mechanisms to manage the MDS for these operations leads to mismatches with the system’s hardware and software architecture properties. To resolve these mismatches, we proposed three alternative configurations for execution of the mapping operation, where each configuration is favored depending on compatibility of requirements and the system properties. Our experimental results confirmed our hypothesis, thereby showing that application-level configuration of multi-dimensional data analysis operations which are constrained to execute on specific sets of resources can help applications exploit the heterogeneity in system architectures to improve performance.

4.2.3 Crossmatch of Multi-dimensional Datasets

Many scientific data analysis applications, especially those that study the variations in observed phenomena over time are characterized by the need to detect changes in distribution of data over large multi-dimensional data spaces. The main goal of these analysis operations is to detect the changes in structure and position of data entities recorded at different epochs in time or other dimensions. For example, astronomers may wish to study the relative change in position of an asteroid belt every few nights. Likewise, a cancer study may involve detecting changes in the
size and coloration of a tumor over time. In general, scientists wish to ask: “how is the recently observed data within a spatial region of a multi-dimensional dataset different from what has been observed in the past for that region?” This change detection involves taking each entity in one dataset and identifying and comparing it against relevant entities in other datasets. This process of cross-correlating entities or objects across multiple datasets is known as object association or crossmatch. The crossmatch problem can be defined as follows: Given two or more datasets of objects, the goal is to find for each and every object in one dataset, all objects in the other datasets that lie within a certain “distance” from the object. Here, the notion of distance is defined based on an application-specific comparison predicate. As the number of observations of data grow in time, so will the number of such comparisons between entities. When objects are compared based on their positions in a common spatial domain, then the problem is referred to as spatial object association or spatial crossmatch. The \(n\)-way spatial crossmatch problem is defined as follows: “Given \(n\) datasets of objects, for each object belonging to one dataset, determine all matching objects from the remaining \(n - 1\) datasets that lie within a search distance of \(\theta\) spatial units from the source object.”

A typical analysis scenario involves newly acquired data (corresponding to a spatial region of the dataset) being crossmatched against all known historical data acquired for that region. Historical datasets are often maintained on disk, where they grow over time as newer observations are recorded. In the first stage of the analysis, known as ‘preparation’, historical data objects that fall within the spatial region of

\[31\] In database parlance, this crossmatch query corresponds to a spatial join operation between relations containing the object tuples.
interest are extracted. In the second stage, the ‘crossmatch’ is carried out via a spatial join of the newly acquired data against the extracted data. In the final (or ‘postprocessing’) stage, the results of the crossmatch are used to update the historical data on disk. The analysis requirements of the crossmatch stage include: (i) for every object in the newly acquired data (or detection), determining the ‘relevant’ objects to compare against, and (ii) pairwise comparison of a detection against all these relevant objects. We present mismatches between the analysis requirements of the crossmatch operation and the architectural properties of the system using an example borrowed from the LSST application (use-case scenario 3 from chapter) workflow.

**Astronomy crossmatch.** The spatial crossmatch is commonly used in astronomy to detect abnormal or transient events occurring in the night sky. Whenever an image of a sky region is acquired by the telescope, the objects extracted from the image (henceforth referred to as DIASource) are compared against all known historical data for that region (henceforth assumed to be stored in a catalog called Object). This is an example of a 2-way spatial crossmatch or a spatial join involving two object collections. Commonly used techniques to carry out the crossmatch stage of the analysis include the Zones algorithm by Gray et al. [49], and the Optimized Zones algorithm by Becla et al. [22].

The Zones algorithm makes use of a spatial indexing scheme known as the zones index. The sky is viewed as a sphere referenced using declination and right ascension coordinates that wrap around near the poles. The zones index bins this sphere data horizontally into non-overlapping “zones” or bands of some predefined height such that objects with similar declination values lie within the same zone. Given such a

---

32 Multi-dimensional data services are employed to perform this extraction efficiently.
partitioning, the search space within which to look for potential object matches for a detection can now be restricted to the containing zone and a small subset of its “neighboring” zones, collectively known as its neighbor set \[65\]. Figure 4.12 shows how the spherical sky is partitioned into zones, and how the zones index can be used to reduce the crossmatch search space.

Figure 4.12: Spatial parameterization of the sky and the use of the zones index

Gray et al. [49] have expressed their crossmatch algorithm in the form of an SQL query as shown in Figure 4.13. Here, \texttt{ZoneNeighbor} is a precomputed table that

\begin{verbatim}
SELECT d.objID, o.objID FROM DIASource d, FOVObject o
INNER JOIN ZoneNeighbor zn ON d.zoneID=zn.zoneID
INNER JOIN o ON zn.matchZoneID=o.zoneID
WHERE o.ra BETWEEN d.ra-zn.\Delta ra AND d.ra+zn.\Delta ra
    AND d.decl BETWEEN d.decl-\theta AND d.decl+\theta
    AND \text{POW}(d.x-o.x,2)+\text{POW}(d.y-o.y,2)+\text{POW}(d.z-o.z,2)<d_{max}
\end{verbatim}

Figure 4.13: SQL query for the \textit{Zones} crossmatch algorithm (DIASource vs. FOVObject)
maintains the neighbor set information for each zone, \( ra \) and \( decl \) are respectively the right ascension and declination coordinates for the object in the sky while \( x \), \( y \) and \( z \) are its projected spatial coordinates that enable fine-grain distance-based comparison. The Optimized Zones (\( OptZones \)) algorithm [22] exploits the application-specific assumption that each zone’s neighbor set contains a maximum of three zones: the containing zone itself and the two “sandwiching” zones (i.e., \( zone_{i-1} \) and \( zone_{i+1} \) for a given \( zone_i \)) along the declination dimension. The neighbor set of a zone is computed on the fly and zone neighbor information need not be precomputed and explicitly maintained like in Zones [65]. Becla et al. have implemented the \( OptZones \) algorithm using SQL and stored procedures supported by certain database systems. Figure 4.14 shows the SQL equivalent of part of the algorithm that does crossmatch for objects within a zone \( i \).

```sql
SELECT d.objID, s.objID FROM DIASource d
INNER JOIN SecondaryZone i AS sz ON d.ra BETWEEN sz.ra-\Delta ra AND sz.ra+\Delta ra
WHERE d.zoneID = i
    AND sz.decl BETWEEN d.decl-\theta AND d.decl+\theta
    AND POW(d.x-sz.x,2)+POW(d.y-sz.y,2)+POW(d.z-sz.z,2)< d_{max}
```

Figure 4.14: SQL crossmatch query for zone \( i \) in the \( OptZones \) algorithm

The size of a zone and the relevance matching information depend on the search radius \( \theta \) specified by the user as part of the crossmatch operation. The Zones algorithm precomputes this information for all zones upfront as a one-time operation whereas \( OptZones \) assumes that it can be computed on-the-fly for each zone. The Zones algorithm involves two expensive join operations with the \( ZoneNeighbor \) table, while the \( OptZones \) algorithm involves one join operation with the \( SecondaryZone \) table.
table created for every zone. The LSST application requires alerts to be generated for previously undetected transient objects within 60 seconds of the image capture. As a result, users impose a time constraint of 30 seconds on the crossmatch stage of the analysis. Traditional systems on uniprocessor configurations are unable to provide the computational power required by the above analysis algorithms to perform several millions of object comparisons within this time constraint.

**Configurations and Mismatches:** The astronomy crossmatch operation must be configured for execution based on the underlying resources. The workflow scheduling strategy used may constrain the crossmatch operation to execute on compute nodes in a shared-nothing hardware architecture. The computational and data movement requirements of the analysis operations are handled by the software architecture present on these resources. Depending on the software architecture, one of configuration decisions for the crossmatch operation involves selecting one among the Zones and OptZones algorithms for a given execution context. The selected algorithm can then be further configured for performance on a system in order to meet the time constraints. We explored the use of two popular software architectures. The first is a *Parallel Database System with Active Disks* that uses hardware acceleration based on active disks for simple database operations to minimize data communication. We used a commercial data warehouse appliance, the *Netezza Performance Server® (NPS)* [9] as a representative for such an architecture. NPS divides query processing among multiple backend nodes known as Snippet Processing Units (SPUs) and employs active disk style processing by delegating database operations such as row-based filtering and column-based projection to pre-programmed Field Programmable Gate Array (FPGA) units placed near the disks on each SPU. The second type of software
architecture could comprise a *High-Throughput Network Database System* that uses the collective memory of all cluster nodes to store data. High-speed interconnects provide rapid access to remote memory in such an architecture. Such an architecture is designed to support the execution of a large number of multiple concurrent queries. We used the open-source *MySQL Cluster* \cite{95} system as the representative for this architecture.

The dataset used in our evaluations was the USNO-B catalog (a public astronomy catalog generated by US Naval Observatory at Flagstaff containing over a billion objects). The crossmatch search radius, $\theta$, was set to 3 arcseconds. We used three different test FOV regions to evaluate crossmatch performance. These regions are characterized by differences in their object density (high, average, low) \cite{65}. The Netezza NPS system representing the first architecture has 56 SPUs, each connected via Gigabit Ethernet switch and equipped with 320 GB local disk with a read bandwidth of 60 MB/sec per SPU disk. Table 4.2 shows the execution time for astronomy crossmatch on NPS assuming that the FOVObject and DIASource are already resident on the SPU disks. We observe that the Zones algorithm performs best on NPS for all test regions in spite of the fact that it involves a pair of expensive join operations performed without index support \cite{65}. This shows that complex query in the Zones

<table>
<thead>
<tr>
<th>Test FOV Region</th>
<th>FOVObject vs. DIASource</th>
<th>DIASource vs. FOVObject</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Naive</td>
<td>Zones</td>
</tr>
<tr>
<td>High density</td>
<td>1950 s</td>
<td>51 s</td>
</tr>
<tr>
<td>Average density</td>
<td>26 s</td>
<td>3 s</td>
</tr>
<tr>
<td>Low density</td>
<td>2 s</td>
<td>0.7 s</td>
</tr>
</tbody>
</table>

Table 4.2: LSST Crossmatch Execution on Parallel Database System

142
The algorithm is being efficiently broken down into snippets for parallel execution on the SPUs, and that the active disk style processing in NPS helps to perform the bulk of the data filtering so that the joins can be performed efficiently on the SPUs even without indexes. The poor performance of the OptZones algorithm on NPS occurs because of the following mismatches: (1) On-demand creation and population of the SecondaryZone; table (prior to performing the join operation between FOV objects within a zone $i$) is expensive and involves extraction of data from disk on all SPUs, creation of a new on-disk SecondaryZone; table distributed across all SPUs, and writing the contents of this table onto disk for each zone. (2) Lack of stored procedures in nzsql means that the SQL statement from figure 4.14 cannot be prepared just once and reified for each zone $i$. Each of the $n$ queries are instead explicitly issued via an external script, thereby entailing parsing and transactional overheads per zone [65].

The MySQL Cluster instance representing the second architecture was set up on 16 nodes of the RII-MEMORY cluster, and configured to contain one manager node, 8 data nodes and upto 8 frontend nodes [95]. A single copy of the data is distributed among the data nodes using the hash-based scheme. Table 4.3 shows execution times for the crossmatch algorithms for the high and average density regions (In these experiments, the crossmatch queries are submitted to a single frontend node). Crossmatch

<table>
<thead>
<tr>
<th>FOV Region</th>
<th>FOVObject vs. DIASource</th>
<th>DIASource vs. Object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Naive</td>
<td>Zones</td>
</tr>
<tr>
<td>High density</td>
<td>11h 22m</td>
<td>11h 58m</td>
</tr>
<tr>
<td>Average density</td>
<td>10m 16s</td>
<td>10m 57s</td>
</tr>
</tbody>
</table>

Table 4.3: LSST Crossmatch execution on a Network Database System

143
performance is worse on MySQL Cluster than on NPS for both algorithms. In MySQL Cluster, join operations are not executed in parallel. Instead, for every join operation, the tables being joined are moved from memory on the data nodes over the network to the memory on the frontend node where the query was submitted. The join is eventually performed on the single frontend node. The OptZones algorithm issues $n$ queries, one for each zone in an FOV, and each of these includes a join operation. Without any data caching at the frontend node, this highly inefficient join mechanism is a mismatch and has high data transfer overheads which lead to poor performance for the OptZones algorithm.

To resolve the mismatch between the crossmatch algorithms and a system like MySQL Cluster, the inefficient join mechanism can be overcome by partitioning the query workload into a large number of sub-queries that can be issued concurrently. In Zones, the single query is partitioned into a set of sub-queries just like a parallel database system such as NPS breaks it up into snippets. In OptZones, the $n$ queries (one corresponding to each zone) are uniformly divided amongst the multiple frontend nodes. Figure 4.15 shows how crossmatch time for the Zones algorithm decreases

![Figure 4.15: Using concurrency to resolve mismatch on MySQL Cluster](image-url)
as we double the number of frontend nodes. In effect, we are exploiting two important properties of such software architectures to resolve the mismatch – (i) the high-throughput nature of network database systems like MySQL allows for concurrent issue and execution of a large number of small queries, and (ii) the associative nature of the astronomy crossmatch operation where the results for an FOV can be obtained by simply combining the results of the crossmatch performed in parallel on individual smaller zones within the FOV. Dividing the query workload among the frontend nodes not only balances the computation but also minimizes the data transfer overheads required for the join operation. The ideal scenario is one where data is transferred exactly once to each frontend. This can be achieved by dividing the query into as many concurrent sub-queries as there are frontend nodes.

Despite processing queries in parallel, and resolving some of the mismatches between the algorithms and the software architecture, the crossmatch for high-density regions cannot be accomplished within the time constraints imposed by the LSST. So, instead of further configuring the application on existing systems, we designed a new hybrid software architecture for shared-nothing cluster systems that combines the best features of the two architectures explored earlier. This architecture consists of a distributed collection of independent database system instances with one instance running per node in the cluster. That is, on each such data node, we have a single, independent database server instance. Our evaluations on this architecture used 16 nodes of the RII-MEMORY cluster with an independent MySQL server instance on each “worker” node, configured to store all data in local memory. An additional node

\[ \text{data nodes and the average density region were used for this evaluation} \]
serves as a master node and coordinates the query execution. Since every database instance has access to only local data (hence the name, independent), the master node partitions query execution such that all join operations directed to a worker node must be performed on local data without any communication with other workers. Data from FOVObject can be partitioned and distributed among nodes using a hash-based scheme. Table 4.4 shows the crossmatch execution times for different regions on this architecture when we simply partition the data among the nodes without any replication. If there are \( n \) nodes, then we refer to this strategy as the \( n \times 1 \) strategy where 1 refers to the number of replicas of each object in the system. Here, prepare

<table>
<thead>
<tr>
<th>FOV region</th>
<th>Prepare time</th>
<th>DIASource transfer time</th>
<th>Query time</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>High density</td>
<td>1.9 s</td>
<td>5.6 s</td>
<td>1.3 s</td>
<td>7 s</td>
</tr>
<tr>
<td>Average density</td>
<td>0.9 s</td>
<td>5 s</td>
<td>0.2 s</td>
<td>5.4 s</td>
</tr>
<tr>
<td>Low density</td>
<td>0.8 s</td>
<td>5 s</td>
<td>0.1 s</td>
<td>5.3 s</td>
</tr>
</tbody>
</table>

Table 4.4: Crossmatch time using 16x1 strategy (OptZones algorithm), FOVObject vs. DIASources

time refers to the time taken by each node to extract FOV objects from Object and load them into FOVObject, DIASource time is time taken to transfer part of or all detections from the master node to the workers, and query time is the actual time to crossmatch data on the workers. The total time also includes the time to load the detections received by a worker into an in-memory DIASource table. By configuring data storage within the application such that any data accesses are only to local memory, the execution time for the crossmatch is reduced so as to fall within the time constraints even for the high-density regions.
Going beyond the LSST application requirements, crossmatch execution can be configured on this hybrid architecture by varying the degree of data replication. A set of \( n \) worker nodes can be divided into \( g \) groups of \( p \) nodes each. Data from the historical object datasets are distributed among the nodes in each group using a hash-based scheme. Here, \( g \) is the replication factor because replicas of data exist in each group, and there are \( g \) such groups. At one extreme, we may have an \( n \times 1 \) grouping, i.e., a single group consisting of \( n \) nodes. This corresponds to the case where the data is uniformly partitioned amongst the nodes without any replication, and each query potentially requires access to data from all nodes. At the other extreme, we can have a \( 1 \times n \) grouping meaning we have \( n \) groups of one node each, where data is replicated across all the nodes, and any query can be executed independently by any node/group. By modifying \( g \) and \( p \) for a given \( n \), we can evaluate intermediary grouping schemes which help us control the data partitioning and replication in a more flexible manner using this architecture. Each combination of \( g \) and \( p \) values used corresponds to a different variant of the crossmatch component for this particular architecture. The amount of data transferred and the computational workload on each worker node will depend upon the grouping strategy adopted. In the \( n \times 1 \) (only partitioning, no replication) variant, every join operation potentially needs data from all nodes on account of the uniform partitioning of the FOVObject table. Hence, the master node will send DIASource to all the data nodes. This will be a bottleneck in cases where the DIASource is extremely large and comparable to the number of objects in the FOV, or in the case where there is a low bandwidth network between the master and the worker nodes. In the \( 1 \times n \) case, each join query can be executed by any one of the nodes. So, the master node divides the query workload (i.e., the
The `DIASource` table is equally distributed among the data nodes and sends only \((1/n)th\) the number of `DIASource` tables to each data node. We are reducing the volume of data communication, but each data node will have to crossmatch its share of the `DIASource` against all objects in the FOV. This case would ideally suit worker nodes with fast processors and slower networks. In the intermediate grouping strategies, the master node will send subsets of `DIASource` of various sizes to groups of data nodes.

![Bar chart showing execution time](chart.png)

Figure 4.16: Varying replication factors and query partitioning mechanisms (Opt-Zones algorithm, high density FOV region, `DIASource` vs. `FOVObject`)

Figure 4.16 shows how varying the data replication factor affects the execution time of the crossmatch for the high density FOV region in the LSST application. As replication factor is increased, the worker nodes spend more time on query execution. This is because each worker crossmatches a smaller set of detections from `DIASource` against a much larger subset of `FOVObject`. The transfer time decreases only marginally. Since the size of `FOVObject` \(\gg\) size of `DIASource` in the LSST test cases, this trend is expected. In cases where the sizes are comparably large, we should
observe a decrease in transfer times with increasing replication factors and a more gradual increase in the query execution times. In this way, the replication factor can be used for application-level configuration of the crossmatch execution for different sizes of DIASource and FOVObject.

Conclusions. Crossmatch of multi-dimensional datasets acquired at different points in time is used to detect changes in the distribution of data. The defining characteristic of this class of operations is the use of spatial join queries to determine relevant matching information for data objects and then performing pairwise comparisons against the matching data. Algorithmic variants for the crossmatch operation may implement different querying styles, some of which may not be compatible from a performance perspective with the underlying software architecture on the system. For the astronomy crossmatch, results show that the Zones algorithm performs better on one architecture (NPS) because OptZones constructs and populates several intermediate tables on disk (as opposed to memory) for every zone in an FOV, and the overhead of high disk I/O leads to its poor performance. On a second architecture (MySQL Cluster), both algorithms perform poorly because joins are not executed in parallel, but instead, on a single frontend node. Performance can be improved by partitioning a query into a set of concurrently issued sub-queries to exploit the high-throughput nature of this architecture. If the nature of the workloads is known in advance, our hybrid architecture (that combines the best features of different architectures) can be used to configure the crossmatch execution to obtain good performance under different conditions. The degree of data replication is used to configure query partitioning and execution on the nodes to meet any time constraints.
4.3 Summary and Discussion

This chapter investigated the interplay between the characteristics of common multi-dimensional data analysis operations and the heterogeneity of the underlying system architectures. We show how different kinds of application-level configurations can help overcome any mismatches between analysis requirements and system properties that may come into play at large data scales. These configurations are implemented as services extending the capabilities of data-centric and application-centric middleware designed to run parallel dataflow programs (Figure 4.17).

Data-Intensive Analysis Applications

Application-level Configuration services
Workflow-level, component-level

DAGMan, Falkon  Pegasus, Swift, Kepler  Taverna, Trident
Workflow Management Systems: Services/Task-based, Abstract/Concrete (Workflow execution & monitoring, High-level language, Provenance support)

Multi-dimensional Data services

ADR, Hadoop/MapReduce  DataCutter, Dryad, IBM System S
Scalable Distributed Dataflow Processing (Object-oriented/Component Frameworks, Language and Runtime Support)

Figure 4.17: Summary of contributions in this chapter (highlighted in red)

The contributions in this chapter sought to help scientists with the following questions in the context of large-scale multi-dimensional data analysis: (i) Given a heterogeneous computing environment with multiple cluster sites comprising different assembly of commodity components, can an application’s execution configuration
exploit the heterogeneity in any way to improve end-to-end performance and to meet time constraints?, and (ii) If workflow scheduling strategies constrain a workflow task to execute on a set of resources, what are the configuration parameters that could influence performance at large data scales? From our experimental evaluations, we observed the following:

- At large data scales, the heterogeneity of computing systems comes into play. Hybrid workflow scheduling strategies that take into account the relative amounts of time spent by each task on computation and I/O (and distribute tasks accordingly) lead to reduced execution times.

- Application-level configurations can overcome mismatches when individual components or ensemble tasks are constrained to execute on a set of resources. To ensure optimal performance, flexible management of data and computations is the key, as is evidenced from three different motifs:

  - In global data structure computations, dependency stalls and communication overheads can be overcome by locally computing only partial data structures and aptly modifying downstream tasks while introducing negligible overheads.

  - For spatial mapping, mapping data structures could be maintained on disk or in memory or computed multiple times depending on the relative speed and capabilities of different system components.

  - For spatial joins, distributed database systems could be configured in different ways with partial replication support that can be adjusted based on data access patterns and dataset characteristics.
Data-intensive scientific applications include feature detection and classification tasks that model human perception of data objects within multi-dimensional datasets. These high-level analysis tasks commonly employ several complex numerical models to identify and understand millions of data objects in macro-scale studies. Earlier chapters described services for high scalability and application-level configuration that eliminate performance bottlenecks for specific operations. These services preserve the quality of the analysis results. However, for high-level analysis, the optimizations provided by these services may not be sufficient to obtain results in reasonable amounts of time. The adaptivity of high-level analysis, wherein tasks can execute at multiple performance levels and produce output of varying quality, can be exploited to meet execution time constraints in such cases. This chapter presents my understanding of accuracy trade-off mechanisms in multi-dimensional data analysis applications, and their use in meeting application-level quality of service requirements put forth by the user. Specifically, I show how scalable services can be constructed to trade accuracy of high-level analysis tasks for performance in application-directed ways.
5.1 Background

Users in various application domains may be willing to sacrifice the quality of output produced by high-level analysis tasks in order to obtain results in lesser time. The limited perceptual ability of scientists to discern minor variations in output at large data scales implies that tasks can get away with producing output that is not completely accurate. However, the users may also wish to exercise control over such accuracy trade-off mechanisms during execution; one should not produce inordinately erroneous results in an attempt to meet the user’s constraints on time. As a result, users may place threshold limits on the accuracy (referred to as application-level quality-of-service constraints) that such analysis operations must satisfy.

**System-level QoS.** The notion of quality-of-service (QoS) requirements and quality-aware systems was first introduced in networking protocols where existing network resources had to scale to bursty traffic workloads. Here, the network resources are reserved for different applications based on their requirements—critical applications such as streaming multimedia and online gaming would be apportioned the desired transmission rates. But, for non-critical applications, network resources including routers and switches reserve the right to drop packets of data. Moreover, some applications may rely on best-effort network protocols like UDP which do not guarantee delivery of packets. The assumption here is that the adaptivity of these applications helps them tolerate any loss of packets that occurs when network capacity is insufficient. Similarly, storage subsystems also support varying levels of QoS to applications when faced with increases in workloads. When the number of query requests grows very large, critical applications may be assigned the desired number of resources so

\[34\text{The benefits of accuracy trade-off mechanisms depend on how well the applications utilize them.}\]
that all their queries are executed at the best quality. On the other hand, for non-critical applications, the subsystem may adopt relaxed consistency for execution on a limited set of resources wherein data writes may not be immediately visible, but only eventually visible to future queries. There is no guarantee that a query has access to the latest copy of the data, or that multiple queries ‘see’ the same copy of the data. Here again, it is assumed that adaptive applications can tolerate such staleness in stored data when the resources are limited.

Packet dropping and relaxed consistency of writes are examples of accuracy trade-off mechanisms or parameters that are employed to improve the system-level performance under heavy workloads. QoS requirements are specified to configure these parameters by placing thresholds on accuracy and performance metrics associated with the execution. Examples of metrics include the error rate, latency, jitter, or packet dropping probability for the network subsystems, and data freshness or availability for the storage subsystems. For instance, when relaxed consistency is used as a trade-off mechanism, a QoS requirement can constrain the data freshness metric such that any data accessed by an application not be over $t$ time units old. This requirement causes the storage subsystem to configure the relaxed consistency parameter in such a way that data is written at least once every $t$ time units. Accuracy-related parameters and metrics such as the ones described make it possible for the adaptivity of application instances to be harnessed in appropriate ways for performance improvements. The parameters and metrics described so far are common to a wide range of adaptive applications and target domains. Hence, it makes sense to incorporate such QoS provisioning at the level of networks or storage subsystems that control the allocation of resources to individual application execution instances. We refer to these
as system-level quality-of-service, since the accuracy trade-off functionality is under the control of the subsystems and not the applications. System-level QoS are used to constrain execution time of analysis, availability of data, confidentiality and integrity of data access, location affinity of execution and other classes of metrics that are not limited to any single application domain.

**Application-level QoS.** Application-level quality-of-service extends the notion of system-level QoS requirements by allowing users to impose additional constraints on domain- or application-specific metrics to exercise fine-grain control over the trade-off mechanisms. Application-level QoS is important from an end-user’s point of view. Scientists in different domains may find it easier to describe the constraints on the output quality using domain-specific terms as opposed to using system-level metrics. Application-specific metrics are more closely concerned with the user-perceived quality of the result or accuracy of the analysis task. For instance, in the context of multimedia and distributed content delivery applications, the end result is typically a rendering of the output data for the user, either in the form of a video or by using an immersive visualization environment. The adaptivity within these applications allows them to render output even when all the necessary data is not available. Users can place application-specific QoS constraints on the precision or clarity of the visual output (e.g., the frame size should be at least $1920 \times 1800$ pixels, or the frame rate must be $\geq 30$ frames per second). A fixed network bandwidth of $b$ bytes per second may be required to guarantee the desired frame rate, but such system-level metrics are dependent on the resources. Application-level configurations are used to satisfy such QoS constraints when the resource level parameters are known. Application-level QoS is typically achieved by configuring parameters that are specific to an application.
instance or to several instances within a particular domain. In the above example, the rendering application can be configured using one or more domain-specific trade-off mechanisms such as: drop intermediate video frames (via unreliable transmission of frames from the server to a client), or filter data within one or more frames, or use lossy compression techniques such as JPEG for individual frames. Each such mechanism or parameter is accomplished through a reconfiguration at the application-level, as opposed to within individual subsystems. Similarly, in numerical methods, applications may adaptively refine the mesh of grid points at regions which require finer resolutions. The grid partitioning and refining techniques are viewed as accuracy parameters to the application whose value settings can affect the computational quality of the result. In service-oriented architectures where application execution is realized through one or more existing services, Service Level Agreements between the users and resource providers ensure that system-level QoS guarantees are met by the services. The adaptive nature of applications could influence the resource provisioning strategies. Thereafter, the configuration of application instances on the allocated resources is used to provide application-level QoS. In summary, system-level QoS involves re-allocation of resources and rescheduling of application components on the resources as the accuracy trade-off parameters to improve resource utilization while meeting thresholds on system-level metrics under heavy workloads; Application-level QoS on the other hand relies on the reconfiguration of analysis tasks to improve performance on a given set of resources while meeting application-specific quality metrics.
5.2 Adaptness of High-level Analysis Tasks

High-level analysis components such as object identification and feature-based classification within many application instances are computationally intensive in nature. For such components, adding more resources to the system does not necessarily translate to improvements in performance especially at large data scales. In section 2.3 adaptivity was identified as one of the important workload characteristics of operations that analyze multi-dimensional datasets. Adaptivity can be defined as the ability of these applications to respond to changes in execution conditions by operating at different performance levels leading to varying levels of correctness or fidelity in the output. This chapter focuses on the adaptivity characteristic of our target application workloads, which can be attributed to one or more factors within an execution context:

1. **Locality causes Redundancy.** Large multi-dimensional datasets exhibit varying degrees of spatial and temporal locality of their feature content. For example, in 3D reconstructions, the basic structure of the entity under study remains consistent across multiple focal planes, thereby exhibiting spatial locality along the Z-dimension. Consequently, for certain kinds of analysis operations that are based only on the structure of the data entities in each 2D plane and not the content of the data, processing data from multiple dimensions (focal planes in the above example) leads to redundancy in computations. By performing the analysis on reduced data (e.g., a single focal plane – the averaged or projected plane) as opposed to on all planes, a large number of redundant computations
can be avoided thereby leading to substantial savings in execution time without degrading the quality of the result.

2. **High-level Analyses have no Perfect Result.** High-level analysis operations such as object segmentation to segregate objects (e.g., light sources from a dark background), feature extraction to extract feature sets for each object, and classification to distinguish between objects based on its feature content are using combinations of increasingly complex numerical models to ensure that analysis leads to accurate results. For such operations, the perfect solution or result may not exist because the techniques employed to model human perception are often statistical in nature, and are always accompanied by confidence metrics or error estimates. Within specific application domains such as medical image registration, efforts to statistically compare results against *gold standards* or *ground truth*, where available, are used to gauge the analysis accuracy. As a result, analysis operations need not always strive to achieve the perfect output. Instead, less accurate outputs may be agreeable to the applications.

3. **Multiple Computational Models.** This factor is more of an extension to the fact that high-level analysis operations employ complex numerical models to process the data. However, this is an important extension because often, the analysis operations are required to determine results based on a large number of models (e.g., the determination of whether a cell is cancerous or not may be based on tens to hundreds of cancer classification models to obtain a consensus view). Not all available models may be applicable under all circumstances. Thus, the ability to eliminate those models that are non-factors in the analysis
could lead to performance improvements while causing a negligible drop in accuracy.

5.2.1 Trade-offs in Multi-dimensional Data Analysis

High-level analysis tasks in data-intensive scientific applications are being increasingly automated in software and are statistical or probabilistic in nature, as is evident from the growing use of data mining and machine learning based techniques to synthesize information from the data. When compared with application-level QoS in general, accuracy trade-offs for such tasks have distinguishing characteristics brought about by the multi-dimensional nature of the datasets and the relatively high spatial resolutions (e.g., several millions of data objects, each having a large set of features) of the data.

1. **Accuracy depends on data content:** Unlike general adaptive applications, the accuracy metric for high-level analysis tasks may not be directly measurable or computable using just the application-level metadata. Instead, the output data may need to be produced first, and then the accuracy is determined from statistical computations based on the content of the output data. To illustrate this point, we compare the accuracy determination for a generic content delivery application against that of a high-level multi-dimensional analysis task. In the former, accuracy metrics such as *frame rate* and *frame size* that represent the expected quality of the rendered output can be directly measured. Moreover, some of these measurements can be realized simply based on application-level metadata without actually rendering the output for the user. If the *audio sampling rate* is one such accuracy metric, one can extract the audio signal alone
from the output and measure the quality without producing the desired output. These accuracy metrics are often combined in specific ways to form a ‘benefit function’ or ‘utility function’ that users wish for the trade-off mechanisms to maximize. The quality of the results in such applications is well-defined as there is little dependence, if any, on actual data content. In the latter, there is a considerable dependence of output quality on data content as high-level analysis tasks are designed to model human perception of the data content.

Accuracy of high-level analysis tasks is highly subjective in nature because no perfect solution may exist for the task – the most accurate solution may vary from user to user. This leads to variability in the quantitative techniques used to determine accuracy of high-level analysis tasks. Consider the segmentation of data into objects as an example of a high-level task; segmentation is responsible for detecting and extracting data entities that can be perceived and identified within a domain. Segmentation has no widely-accepted perfect solution. So, the accuracy is determined using either confidence methods, goodness methods or discrepancy methods. In confidence methods, the analysis tasks are assumed to be equipped with built-in techniques that can quantify the output quality through an accuracy metric. Such metrics may express the confidence that the analysis algorithm has in the quality of the output it produced. In image analysis, a segmentation algorithm operating on an image at its highest resolution may be able to identify and segment circular and elliptical shaped data entities with close to hundred percent confidence. However, at lower data resolutions, the algorithm may not be as certain about the identified object, and hence its output is associated with lesser confidence (e.g., only sixty percent likelihood
that the detected data object has a well-defined elliptical shape). Goodness methods are based on statistical measurements over the regions segmented by the algorithm. Once the output data is produced, measurements of specific properties such as overlap between segmented regions, intra-object uniformity, inter-object contrast, entropy, edge quality and continuity may be combined in statistically meaningful ways to produce one or more accuracy metrics for the segmentation. Here, the computation of output and the accuracy metric are decoupled; the analysis task is not responsible for producing any accuracy metric. Finally, in certain application domains, especially those that analyze image data, discrepancy methods exist that assume the presence of a reference output to compare against known as ground truth or the gold standard result. In segmentation, the ground truth could be a set of objects identified offline by a human expert. Accuracy metrics are then obtained based on the various discrepancy criteria between the observed output image and the ground truth. Regardless of the method employed, every time an accuracy parameter value is changed, the output has to be produced first, and only then is the accuracy metric determined based on the data content.

2. **Accuracy parameters for high-resolution data:** Reconfiguration mechanisms for application-level QoS have traditionally been centered around preserving the data flow within an application, and reducing the amount of computation performed on the data. One common accuracy parameter used for adaptive analysis tasks is the substitution of one algorithm for another that may analyze the same input data in computationally less-intensive ways to produce output of a lesser quality. In analysis tasks that are characterized by iterative processing
over the data, the number of iterations can be reduced to produce faster, less accurate results. When the analysis involves applying a high degree polynomial function over the data, the degree of the polynomial can be bounded such that the task is guaranteed to produce some lower quality output within a certain time deadline.

Large scientific datasets have very high spatial resolutions and may contain several millions of data entities. It may not be possible for high-level analysis tasks to iterate over such high resolution data and also meet application-level QoS within reasonable time by simply reducing the amount of computation done per data entity. For our target applications, non dataflow preserving accuracy parameters that reduce the amount of data processed by the analysis tasks are commonly used for application-level QoS. The effect of such parameters is to filter the input data to an analysis task either prior to or during execution, thereby reducing the data overheads and the resource requirements of the computation. Several data approximation techniques exist for spatial, multi-dimensional data, some of which may be common across multiple domains:

(a) Data models or data representation techniques employed by the analysis are parameters to control the accuracy and execution time. If an analysis task must determine the structural similarity between two or more spatial data entities with complex shapes (such as proteins with tertiary structures), shape approximation techniques (such as minimum-bounding

\[35\text{In general, the lesser the amount of data that needs to be processed, the lower are the resource requirements, and the lesser is the execution time for the analysis.}\]
polygons/spheres or shapelets) can be employed to reduce the amount of computation required. The closer the modeling is to reality, the better the output quality; hence, accuracy drops when approximations are used. Similarly, for certain types of analysis operations, one can transform input data into a different domain (e.g., wavelet transformations) and perform a reduced set of appropriate computations in the transformed domain to obtain faster results, albeit of different quality.

(b) Shape simplification or generalization is a commonly used mechanism of reducing the number of points needed to represent a line or a polygonal region in the multi-dimensional space. Shape simplification may not yield substantial performance improvements on a per computation basis. However, when considered in the context of indexing or matching of a data entity against millions of other data entities, the cumulative improvements in execution time are beneficial.

(c) Sampling the data space to process only a small subset of the data is a highly common data approximation technique. This is similar to the packet-dropping parameter in system-level QoS. Sampling has been used in database systems to speed up query execution and index lookup. The samples may be chosen at random or systematically. Sample generation techniques may vary ranging from reservoir sampling to biased sampling to sliding-window sampling. Scientific data can often be processed at multiple resolutions, where lower resolutions contain lesser data than the higher resolutions. Tessellation groups together spatially proximate entities in order to produce a coarser granularity representation of the data.
(d) Certain data models especially in the domain of geographic information systems use multiple segmentation scales, wherein, an object with a well-defined shape at a higher scale may be represented using a point at a lower scale.

3. **Accuracy trade-offs for spatial, multi-dimensional data:** For large multi-dimensional data analysis, accuracy metrics and parameters could be associated with different dimensions or features of the data and also data entities at different granularities. In color-based classification of image data, confidence values may be computed based on the distribution of color intensity values within data objects. In shape-based classification, confidence values are computed based on the edge quality, entropy and other high-level features. Depending on the analysis requirements, accuracy metrics could be associated with individual data entities or with specific regions of a dataset, or with entire datasets. For example, shape-based classification tasks may compute confidence in classification at the granularity of individual data objects, while other classification tasks may do so at the granularity of regions comprising multiple objects. The overall accuracy metric for the analysis task may be obtained by aggregating the accuracy values at lower granularities in statistically meaningful ways. When the overall accuracy for a dataset is computed this way by aggregating the accuracies for individual data entities or regions, then one need not wait until the entire dataset is processed to determine the overall accuracy metric. Instead, partial accuracy metric values can be computed at all points of time over the execution, based on those portions of the dataset that have been processed until then. This mode of computing accuracy metrics at a higher granularity from
those at lower granularities can help in dynamic decision-making to favorably select portions of the dataset to process.

We illustrate these characteristics using an example use-case scenario borrowed from the domain of biomedical imaging in the context of cancer prognosis.

**Use-case scenario 5: Neuroblastoma Classification.** Neuroblastoma is the most common extra-cranial, solid, malignant tumor that affects children. It is one of the most poorly treated cancers that considerably degrade children’s health. These aspects of neuroblastoma have prompted research on its pathological characteristics and neuroblastoma tumor classification protocols. Tumor differentiation is a measure that indicates the similarity of a tumor to the corresponding normal tissue architecture. Better differentiated tumors usually bear less aggressiveness, and thus having better chance to be cured. As a result, an accurate grading of a neuroblastoma sample is crucial to pathologists before an appropriate treatment plan can be made to a particular case. Figure 5.1 shows a multi-resolution based neuroblastoma classification workflow [59] (developed by researchers at the Ohio State University) based on computer vision and pattern recognition methods and applied to images from high-power light microscopy scanners. The inputs to the workflow are compressed chunks from whole-slide images of tissue samples, digitized at 40x magnification by a scanner. Before any analysis of images, a layered multi-resolution hierarchy of chunks is created and stored on disk; each image chunk is down-sampled in such a way that the loss of frequency information is minimized [59]. Each chunk is processed independently, using a sequence of segmentation, feature construction, feature selection, and classification steps. The classification step is based on the International Neuroblastoma Prognosis Classification Scheme [100], and additionally produces a degree
Figure 5.1: Data analysis workflow for Neuroblastoma classification

*of confidence* measure based on how likely the classification of the chunk as computed by the analysis approach is correct. Thus, this application uses the confidence method to compute an accuracy metric. The confidence value is obtained by taking a weighted mean of consensus votes made by eight different algorithms employed for the classification task.

The accuracy parameters we use for trade-off mechanisms in this application are the ‘resolution’ and ‘processing order’ of image chunks. An adaptive multi-resolution strategy is employed for classification of the chunks. Each chunk can be classified at multiple resolutions, i.e. all steps in the workflow can be applied to chunk data at multiple resolutions. The processing of a chunk starts at the lowest resolution, and proceeds in an iterative manner onto higher resolutions. For application-level QoS, users can specify a minimum threshold value on the confidence of classification of
each chunk in the image. In other words, the degree of confidence for classification should exceed the user-defined threshold value for the classification to be acceptable. If the confidence value at a given resolution $j$ is below the user-defined threshold $x_j$ at that resolution, the chunk is processed at the next higher resolution. If the confidence value at a given resolution exceeds the user-defined threshold, the chunk is considered to be satisfactorily classified. This decision rule controls the transition process from the lower to higher resolution levels and is based on the resulting confidence measure computed by the classification step. The lowest resolution $j$ of chunk $C$ for which the confidence level $a_j$ exceeds the corresponding threshold $x_j$ is referred to here as the target resolution of $C$. Once the target resolution of a chunk has been reached, the chunk is said to be “finalized” and need not be processed at the remaining higher resolutions. A chunk processed at the highest resolution is always finalized – i.e., for a given chunk, the value of the accuracy metric is highest at the maximum resolution. The goal is to classify all image chunks with an acceptable degree of confidence. In this application, users may specify one or more constraints and QoS requirements such as the following:

- Maximize the average accuracy (average confidence level) of classification for the image within $t$ time units.
- Maximize the number of chunks that are finalized within $t$ units of time.
- Maximize average confidence in classification of individual data entities within $t$ time units.

$^{36}$Users may specify different threshold values for the different resolutions at which a chunk is processed.
The multi-resolution processing strategy provides data approximation within a region or chunk. For time-constrained QoS requirements, we can additionally perform coarse-grain sampling across chunks, i.e., to maximize accuracy within a time constraint, we can selectively process only a subset of favorable chunks from the image at their target resolutions. This selection of chunks may be done in specific order that is determined dynamically depending on the analysis requirement, and on the accuracy metric values produced for individual chunks. We refer to this trade-off strategy as the processing order parameter that configures the order in which data chunks are selected for execution. In this particular application instance, accuracy metric values are produced at the granularity of an image chunk, and the accuracy of analysis for the overall image is obtained by aggregating these values depending on the user requirement. If the user wishes to maximize the average confidence across all image regions, then the mean of individual accuracy metrics is computed. If the user wishes to maximize the number of accurately classified image regions, then the sum of all finalized chunks is computed. In this use-case scenario, we have only one accuracy metric associated with the application (confidence value computed for the classification component). In general, we could have multiple accuracy metrics associated with different components.

5.2.2 Modeling Accuracy-Performance Trade-offs

The relationship between accuracy parameters and the metrics (both quality-related metrics and execution time) is not always straightforward and depends on several factors within an application’s execution context. In this section, we show how this relationship could be modeled and evaluated using simulation-based studies. As
an example, consider image content classification that can be carried out at multiple data resolutions. Table 5.1 illustrates the variation in performance and accuracy of classification as the resolution of the image is varied. Here, the processing time is the overall time taken to process all chunks of the image at the given resolution, and the confidence is the average of the confidence in classification values obtained for all chunks in the image at that resolution. The execution time for an image at the lowest resolution is much lesser than at the highest resolution. As we increase the resolution (i.e., reduce the input data approximation), the accuracy improves whereas the execution time always reduces (performance improves). However, this trend may not hold true all the time, and in general, the relationship between accuracy parameters and the metrics is difficult to estimate before execution.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Processing time</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 × 64</td>
<td>374 s</td>
<td>0.446</td>
</tr>
<tr>
<td>128 × 128</td>
<td>483 s</td>
<td>0.483</td>
</tr>
<tr>
<td>256 × 256</td>
<td>1088 s</td>
<td>0.509</td>
</tr>
</tbody>
</table>

Table 5.1: Performance and accuracy at varying image resolutions.

**Simulation under Ideal Conditions.** We present our simulation studies which model the execution of an application instance such as the Neuroblastoma classification workflow (use-case scenario 5). Specifically, we compare a basic execution strategy that does not configure any accuracy parameters against ideal-condition strategies which configure the parameters in accordance with estimates of the impact of configuration on the metrics and execution times. That is, the simulated strategies work under the assumption that accuracy of analysis can be computed for different parameter value settings without processing the data. The difference between the
performance of a basic strategy and a strategy that is precisely cognizant of the effects of configuring parameters will give us the scope for improvement over the basic strategy for execution under non-ideal conditions. As an example simulation study, consider the resolution of data chunks, and the processing order parameters from the above application. When the accuracy and performance characteristics based on these parameters are not known a priori, a chunk has to be processed until its target resolution \( k \) is reached. One basic approach is to choose a chunk \( T_i \) at random at a time and process the chunk iteratively starting from the lowest resolution instance until the target resolution is reached. Once \( T_i \) is finalized, the next task from \( S \) is chosen at random. This strategy assumes that the accuracy metric values and execution time are not known a priori. The main idea behind the basic strategy is to avoid starting from a region of the image where all chunks have to be processed at high resolution to be finalized or the confidence level of classification per chunk increases very slowly as the chunk resolution is increased. This approach is illustrated in Figure 5.2 and is referred to as \( R A N D \) (for Random). Under ideal conditions, we assume that the target resolution at which a chunk must be processed is known beforehand. Thus, in the first ideal strategy called \( T A R G E T - O N L Y \), each chunk is not processed iteratively, but only at its target resolution. In the presence of time constraints, the chunks can be processed in specific orders to maximize the benefit to the user. For example, if the user wishes to maximize number of chunks processed, \( M A X T A S K \) orders the chunks in such a way that chunks that are finalized at lower resolutions are processed first ahead of other chunks. Similarly, if the user wishes to maximize the average confidence in classification of all data entities, \( M A X A C C \) orders the entities for execution in such a way that those that yield highest increase in
**Input:**
Set $S$ of chunks
Set $X$ of thresholds

**Algorithm:**
1. loop until $S$ is empty or time limit is reached
2. Select a $T_i$ randomly from $S$
3. $j \leftarrow 1$ (* lowest resolution instance of $T_i$ *)
4. loop until $T_i$ is finalized or time limit is reached
   5. Process instance $j$ of $T_i$
   6. if $a_{ij} \geq x_j$
      7. $T_i$ is finalized
   8. else
      9. $j \leftarrow j + 1$
   10. endif
11. endloop
12. endloop

Figure 5.2: Basic strategy: No parameter configuration (*RAND*)

accuracy relative to the execution time at their target resolution are processed first. *EPOCH* is a variant of *MAXACC* that tries to ensure fairness to all data entities. Our simulation studies were based on 100 data entities or chunks which could be processed at up to 10 resolutions. In the simulations, the execution time and confidence in classification are set to increase linearly with increase in resolution. Hence, the accuracy and performance characteristics in the initial set of studies are said to be linearly proportionate. Figure 5.3(a) shows the gap between *TARGET-ONLY*, the strategy for ideal conditions and *RAND*, the basic strategy. It also shows that the configuration of accuracy parameters like chunk resolution and order of processing can help bridge this gap. Figure 5.3(b) shows a huge improvement in the average confidence per task when one uses the *MAXACC* or *EPOCH* strategies. The performances of *MAXACC* and *EPOCH* were comparable in this case.
Figure 5.3: Simulation studies for accuracy parameter configuration. 100 tasks, 10 resolutions per task

Variations under Non-ideal conditions. There are important variations that one must model in the simulations to account for possibilities that arise in real-world applications: (i) Linear proportionality may not always be valid. In certain cases, the execution time may increase rapidly at lower parameter settings, while the accuracy metric value may exhibit a more gradual increase with resolution. On the other hand, the accuracy of analysis may grow steeply with increasing resolution without considerable increase in execution time. The simulation results show that regardless of the relationship between accuracy parameters and the accuracy metrics and performance, parameters can be configured in specific ways to provide improved system response over a basic strategy that does not configure any parameters. (ii) The second variation is one where the relationship between an accuracy metric like confidence and an accuracy parameter like data resolution cannot always be assumed to be a non-decreasing, convex function. Monotonicity implies a conventional trade-off mechanism where a loss in accuracy always leads to an improvement in performance. On the other hand, we could also have a non-monotone relationship between the parameter and the
accuracy of analysis, where an increase/decrease in parameter value does not always ensure a corresponding increase/decrease in the accuracy. Such a relationship can be observed in certain forms of data analysis, where users prefer to classify entire regions or chunks of data based on their content as opposed to classifying individual data entities. This non-monotonicity implies that a non-straightforward trade-off mechanism, because a decrease in performance does not always result from an increase in output quality. This kind of relationship makes it difficult to predict the performance/accuracy of analysis that is required in order to meet the user QoS requirements.

**Summary.** Our simulation studies model the relationship between accuracy parameters and the performance and accuracy of analysis based on the requirements of the classification use-case scenario. Similar simulation studies can be undertaken to identify relationships between different sets of parameters and metrics. Strategies like `TARGET-ONLY` use ideal conditions (precise estimates of accuracy mechanisms) to establish an upper bound on the improvements that could potentially be achieved.
over basic strategies like RAND. However, since accuracy is heavily dependent on data content within our target application domain, we need strategies that will work under non-ideal conditions and realize similar benefits. The MAXTASK and MAX-ACC simulation strategies provide guidance with respect to the likely performance benefits under non-ideal conditions, where precise knowledge of accuracy mechanisms are not available. The non-ideal condition strategies may also need to account for variations in the relationship between accuracy and performance such as non-linear proportionality and non-monotonicity.

5.3 Trade-off Heuristics for High-level Analysis

Based on our experiences with simulation studies, we developed two heuristic-based accuracy trade-off approaches for non-ideal execution conditions to support application-level QoS requirements in high-level analysis applications such as Neuroblastoma Classification. The first is an object-based heuristic, in which the processing order and resolution for individual data entities being classified within the dataset is determined based on accuracy/performance estimates for each individual entity. The second is a region-based heuristic where parameters are configured based on estimates for image regions containing many entities. In this section, we describe these heuristics in detail and provide results from our experimental evaluations using large microscopy image data.

5.3.1 Object-based vs. Region-based heuristics

Object-based heuristics are used when the accuracy metric of interest to the user is computed at the granularity of individual data entities or objects within the dataset.
For example, if the user wishes to classify each individual cells or nuclei in an image with high confidence, then this heuristic can determine the favorable objects to operate on given the time constraints. This heuristic uses a priority queue data structure to determine the order in which the image data objects are processed. An entry in this queue corresponds to an object in the image and is an instance of the QueueEntry class shown in Figure 5.5. Here, data object represents a perceivable

```c++

class QueueEntry {
    data object
    resolution
    confidence
    proc time
    priority
};
```

Figure 5.5: The QueueEntry class

object within the image space, resolution is the resolution at which the data object has been processed, confidence represents the degree of confidence with which the object was classified at resolution, and proc time is the execution time for processing the object at resolution. The heuristic works in two phases: The first (static) phase is one where each object is processed at the lowest resolution. If an object is not finalized at the lowest resolution, an entry corresponding to this object is created in the queue. At the end of the static phase, the queue contains all objects that were not finalized at the lowest resolution. The subsequent (dynamic) phase proceeds as follows: The entry \( q \) at the top of the queue is removed and the corresponding object
A data object is processed at \( q \text{.resolution} + 1 \). If the object is not finalized, the object is re-inserted to the queue at \( \text{resolution} + 1 \). This phase continues until all the objects are finalized or the time limit expires. This phase of the heuristic is dynamic because the deletions and insertions of queue entries are interleaved along with the processing of the objects and the queue is dynamically updated. At any point in time during the dynamic phase, there is at most one queue entry corresponding to any given object. The absence of a queue entry for an object implies that the object is either currently being processed or has been finalized. The heuristic is flexible enough to be applied to different kinds of QoS requirements by simply modifying the queue insertion semantics. We employ the following two insertion schemes to respectively address the “maximize the number of objects finalized” and “maximize the average accuracy” QoS requirements:

1. Given two queue entries, each of which corresponds to a different object, the entry whose confidence value is closer to the user defined threshold at \( \text{resolution} \) has a better chance of exceeding the threshold at the next higher resolution (i.e., \( \text{resolution} + 1 \)). This entry is placed higher in the queue than the other entry.

2. Given two queue entries, each corresponding to a different object, the entry that achieves the highest increase in confidence value per unit of processing time between \( \text{resolution}-1 \) and \( \text{resolution} \) has a better chance of achieving high gain in confidence value per unit of processing time at the next higher resolution. This entry is placed higher in the queue than the other entry.
For application-level QoS: “Maximize the number of finalized objects”, we use the first insertion scheme to give priority to the object that has greater chance of getting finalized by moving it towards the top of the queue. For application-level QoS: “Maximize the average accuracy”, we use the second insertion scheme to give higher priority to the object that promises greater increase in accuracy at the next higher resolution. These insertion semantics are developed based on the assumption that the relationship between the resolution parameter and the accuracy is a monotonically increasing function. However, for the Neuroblastoma classification workflow, users seek to classify image regions or chunks and not individual objects. Our evaluations on image data using the object-based heuristic revealed that the confidence in classification of chunks was not strictly non-decreasing with increasing resolution. As a result, the object-based heuristic does not exhibit the desired improvement over a basic strategy in which no accuracy parameters are configured (Figure 5.6). The number of finalized chunks at any given point of time was not much higher than that achieved by the basic strategy. This uncertainty in the analysis accuracy at different

![Graph showing number of finalized chunks versus processing time for different strategies](image)

Figure 5.6: Object-based heuristic is not useful for application-level QoS requirements at the granularity of chunks or regions (Neuroblastoma classification)
resolutions makes it almost impossible to estimate the behavior of a single chunk in isolation, as data resolution is changed. In order to alleviate this problem, we devised a region-based heuristic, which is described next.

Region-based heuristics are based on the assumption that there will be a high positive correlation between spatially proximate regions in a dataset, because such regions are likely to have high similarity in their feature content. As a consequence, if we determine a region within the dataset that has favorable performance/accuracy behavior for a given QoS requirement, then other regions in the spatial neighborhood of this region are likely to exhibit similar behavior. This heuristic also uses a priority queue with similar queue entry structure as in the object-based heuristic and employs similar queue insertion schemes for different application-level QoS requirements. However, in this heuristic, queue entries correspond to the regions of an image rather than individual data objects or entities. When multi-dimensional datasets are partitioned into chunks for analysis, each chunk can be considered to be a region for this heuristic. For the Neuroblastoma classification application, the region-based heuristic works as follows: Initially, the image is partitioned into $M$ disjoint rectangular regions. Each region has a bounding box and contains a set of chunks that are spatially close to each other and lie within this box. The queue entry structure has an additional element $region$, which points to the corresponding region of the image. In the static phase of the heuristic, a chunk is selected at random from each region. The selected chunk is referred to as the representative chunk for the region. Each selected chunk is processed iteratively, starting from the lowest resolution, until the chunk is finalized. For a given chunk, a queue entry is created; $region$ points to the region that contains the selected chunk, $chunk$ is the selected chunk with $confidence$
value at \textit{resolution}, which is the resolution at which the chunk has been finalized. The entries are inserted into the queue using one of the insertion schemes described in the object-based heuristic. During the dynamic phase, the entry at the top of the queue is selected, and a new chunk is selected at random from the corresponding \textit{region}. The new chunk is processed iteratively, until it is finalized. The region is divided into two sub-regions by the median line between the newly selected chunk and the old representative chunk along the X or Y dimension. In this way, one sub-region contains the newly selected chunk and the other sub-region has the old representative chunk. The other chunks in the region are assigned to the sub-regions based on whether they are to the left (below) or right (above) of the median line. Queue entries are created for each sub-region and are inserted into the queue using one of the insertion schemes. When a region is empty, it is deleted from the queue. The dynamic phase proceeds in this manner until either the queue becomes empty or the time limit expires.

\section{5.3.2 Parallel Execution Strategies}

We describe our efforts at parallelizing the above trade-off heuristics and related runtime optimizations. To carry out the high-level analysis tasks in parallel on the nodes of a cluster system, we use one or more application filters in our OCVM middleware. A master console filter running on one of the nodes determines the order in which data regions or objects within the dataset must be processed and distributes the workload amongst the application filters accordingly. We used a \textit{demand-driven} execution strategy as it enables us to dynamically adapt to changes in load on the workers and changes in network traffic. The console filter uses the algorithm shown in Figure 5.7 to carry out the accuracy trade-off mechanisms in parallel.
Input:
Set of Image chunks $I$
Initial partitioning $X, Y$
User requirement

Algorithm:
1. $I \leftarrow I - \{\text{Background chunks}\}$
2. $requestQ \leftarrow \{\text{initial requests}\}$
3. $chunksdone \leftarrow 0$
4. $regionQ \leftarrow NULL$
5. Partition Image into $X \times Y$ regions
6. Insert each region into $regionQ$
7. while $chunksdone < |I|$
7.1 if ($requestQ$ not empty) and ($regionQ$ not empty)
7.2 then
7.2.1 node $\leftarrow$ Top of $requestQ$
7.2.2 $R \leftarrow$ Top of $regionQ$
7.2.3 $T \leftarrow$ pick a chunk that lies in region $R$ at random
7.2.4 if $R$ has a previously processed chunk
7.2.5 then
7.2.5.1 $\{R1, R2\} \leftarrow$ Partition $R$ into two regions
7.2.5.2 $R1.confidence \leftarrow R2.confidence \leftarrow R.confidence$
7.2.5.3 $R1.time \leftarrow R2.time \leftarrow R.time$
7.2.5.4 Remove $R$ from $regionQ$
7.2.5.5 Insert $R1$ and $R2$ into $regionQ$
7.2.5.6 $R \leftarrow$ Top of $regionQ$
7.2.6 endif
7.2.7 Remove $R$ from $regionQ$
7.2.8 Read the representative chunk $T$ for $R$
7.2.9 Send $T$ to node
7.2.10 Remove node from $requestQ$
7.3 else
7.3.1 Wait for response from any node
7.3.2 Update region with confidence and time
7.3.3 Insert responding node into $requestQ$
7.3.4 $chunksdone \leftarrow chunksdone + 1$
7.4 endif
8 endwhile

Figure 5.7: Console filter scheduler algorithm (Region-based heuristic)
5.3.3 Experimental Evaluation

We present an experimental evaluation of the trade-off heuristics developed for different QoS requirements. Our experiments were carried out on 32 compute nodes of the RII-MEMORY cluster. The dataset used for these experiments consists of four images as described in Table 5.2. These images differ in their data volume(size), content(grade) and the number of non-background chunks. Here, grade refers to the grading of neuroblastic differentiation [60] and can be one of undifferentiated (UD), poorly-differentiated (PD) or differentiating (D). In addition to different QoS requirements and multiple user-defined “confidence in classification” threshold values, these variations give us a wide range of experimental conditions to test the effectiveness of our heuristics. We first test our hypothesis that the region-based heuristic, unlike the object-based heuristic, yields a better response to QoS requirements in Neuroblastoma classification than the basic strategy. For this set of experiments, we considered images I1, I2 and I3. The images are partitioned into chunks of size 512×512 pixels each.

User requirement: Maximize number of finalized chunks within t units of time.

Figure 5.8(a) shows the result for I1 for the case when the user threshold was set to

<table>
<thead>
<tr>
<th>Image</th>
<th>Grade</th>
<th>Data Size (GB) (uncompressed)</th>
<th>Number of chunks</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>UD</td>
<td>11.6</td>
<td>4075</td>
</tr>
<tr>
<td>I2</td>
<td>UD</td>
<td>8.4</td>
<td>7509</td>
</tr>
<tr>
<td>I3</td>
<td>PD</td>
<td>21.7</td>
<td>10519</td>
</tr>
<tr>
<td>I4</td>
<td>D</td>
<td>21.5</td>
<td>16011</td>
</tr>
</tbody>
</table>

Table 5.2: Dataset characteristics

37 Background chunks for each image are eliminated in a preprocessing step and only the remaining chunks are retained for classification.
0.5 at all four resolutions. This experiment was conducted across 16 worker nodes. As

![Graph](image)

(a) Number of finalized chunks as a function of processing time. (16 workers, confidence threshold 0.5)

(b) Percentage increase in number of finalized chunks as a function of processing timer. (32 nodes, threshold 0.7)

Figure 5.8: Benefits of region-based heuristic for application-level QoS: “Maximize number of finalized regions in time $t$”

is seen in the figure, at any given time-step during the processing, our region-based heuristic classifies more chunks than the basic strategy. For instance, if a user had just 2500 seconds to process image $I_1$ with a threshold of 0.5, our heuristic would be able to finalize 3166 chunks, while the basic scheme would result in only 2920 chunks being finalized. The difference in the number of finalized chunks between the two schemes peaked at almost 350. Similar trends were observed when the threshold value for confidence was varied from 0.5 to 0.9. In general, for the extreme cases (very low and high threshold values), one can expect our heuristic to provide little improvement over the basic scheme. This is because very low (high) threshold values would force a majority of the chunks in the image to be finalized at the lowest (highest) resolution. For a given resolution, the processing times for different chunks do not vary much. Hence, if most chunks are finalized at a single low (high) resolution, the order in which
the chunks are processed may not influence the overall performance. Figure 5.8(b) shows the percentage increase in the number of classified chunks that the region-based heuristic achieves over the basic strategy as a function of processing time. The user-defined threshold in this case was set to 0.7 at all resolutions. Experiments were conducted using 32 worker nodes. Results are shown for images $I_1$, $I_2$ and $I_3$. For example, after 10% of the overall processing time for image $I_1$, the heuristic has classified 44% more chunks than the basic scheme. The improvement drops to 5% when 75% of the processing time has elapsed. For image $I_2$, the corresponding improvements are not as substantial as that of $I_1$, while for image $I_3$, we perform no better than the basic strategy at this threshold of 0.7. A plausible explanation for this difference in behavior across images is based on our observations in Table 5.3. Here, $r_1$ is the lowest resolution while $r_4$ is the highest resolution at which an image chunk can be processed. The table gives us the percentage of chunks for each image that are finalized at a given resolution. For example, for image $I_1$ that has 4075 chunks, 26% of these chunks were finalized at the lowest resolution, while 44% had to be processed at the highest resolution in order to satisfy the user threshold of 0.7. At this threshold, image $I_1$ exhibits a good balance between the number of chunks that are finalized at lower and higher resolutions. However, for image $I_3$, 75% of

<table>
<thead>
<tr>
<th>Image</th>
<th>% of chunks finalized at resolution:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$r_1$</td>
</tr>
<tr>
<td>$I_1$</td>
<td>25.9</td>
</tr>
<tr>
<td>$I_2$</td>
<td>18.1</td>
</tr>
<tr>
<td>$I_3$</td>
<td>10.5</td>
</tr>
<tr>
<td>$I_4$</td>
<td>5.1</td>
</tr>
</tbody>
</table>

Table 5.3: Influence of image content at threshold 0.7
the chunks need to be processed at the highest resolution in order to satisfy the user threshold. One must note that these relative percentage figures are contingent on the threshold value. Regardless of image content, lower (higher) the threshold, greater the fraction of chunks that are finalized at lower (higher) resolutions. Hence, the effect is the same as that of specifying an extreme threshold value. When such a large majority of chunks need to processed at the lowest or highest resolution, the efficiency of the heuristic reduces as explained earlier. This difference in behavior of the region-based heuristic across images can be attributed to the influence of image content. Images that are graded differently exhibit differences in the resolutions at which their chunks are classified. This makes a strong case for incorporating factors such as image content into the trade-off heuristics.

**User requirement:** *Maximize average confidence within $t$ units of time.* Figure 5.9(a) shows the result for $I_1$ for the case when the user threshold was set to 0.7 at all resolutions. This experiment was conducted across 16 worker nodes. At any

![Diagram](image)

(a) Average confidence as a function of processing time (16 workers, threshold 0.7)

Figure 5.9: Benefits of region-based heuristic for application-level QoS: “Maximize average confidence across all regions in time $t$”
time during the processing of chunks, the region-based heuristic returns a higher average confidence value than the basic strategy. This increase in confidence is observed to be as high as 9% at one particular instant. The reason we observe this trend is because the region-based heuristic favors chunks that will potentially contribute more to the overall average. The higher the correlation between the features of spatially neighboring chunks is, the better it will perform. Similar trends were observed when the threshold value was varied from 0.5 to 0.9. Figure 5.9(b) shows the percentage increase in the average confidence that the heuristic achieves over the basic strategy at each instant of the processing of the image. The user-defined threshold in this case was set to 0.7 at all resolutions. The results are shown for images I1, I2 and I3. Similar to our earlier discussion, the behavior is different across the three images and this can be attributed to the influence of the image content.

**Influence of threshold:** Figure 5.10 shows the relative improvements of the region-based heuristic over the basic strategy at different confidence threshold values specified by the user. This experiment was conducted for image I3 using 32 worker nodes.

Figure 5.10: Percentage increase in average confidence with varying confidence thresholds, 32 workers

185
and the application-level QoS requirement was to maximize average confidence within time \( t \). At low thresholds (thresh = 0.1 and 0.35), our heuristic was able to yield an increase of upto 25% in the average confidence over the basic scheme. However, at a higher threshold (thresh = 0.7), the observed performance is not much better.

In addition to demonstrating the ability to trade accuracy for performance improvements, an important requirement of the heuristics is that they must scale to handle several millions of data objects and regions efficiently. The parallel implementation must not introduce significant overheads on the console filter as the size of the data or the number of workers increases.

**Scaling the number of worker nodes:** In this experiment, we process the same image \( I_1 \) for the second QoS requirement with a user threshold of 0.7. The number of worker nodes used is varied from 12 through 32. Figure 5.11(a) shows that the parallel implementation of the heuristic scales linearly as the number of nodes is increased. As we double the number of nodes, it can achieve the same average confidence in roughly half the time.

![Figure 5.11: Scalability tests for accuracy trade-off heuristics](image)

(a) Time taken to achieve the average confidence (b) Scaling image data size, 32 workers, threshold along x-axis. Scaling number of worker nodes, 0.8 threshold 0.7

Figure 5.11: Scalability tests for accuracy trade-off heuristics
Scaling the image size. For this set of experiments, we considered images $I_1$ and $I_4$. The experiments were conducted with a threshold of 0.8 at all resolutions and with 32 worker nodes. The QoS requirement is to maximize the average confidence across all the image chunks within time $t$. From Figure 5.11(b) we observe that for a given image, if we process twice the number of chunks, then the time taken increases by roughly a factor of 2. Across different images, we do not observe a strictly linear scaling. Note that the amount of data to be processed in $I_4$ is about 4 times that of $I_1$. However, the time taken to process a data in $I_4$ is greater than 4 times the time taken to process an equivalent amount of data in $I_1$. This is because chunks in different images are finalized at different resolutions. As seen in Table 5.3, 82% of chunks of $I_4$ are finalized at the highest resolution, while the corresponding number for $I_1$ is just 44%. Hence, the time taken to process all chunks of $I_1$ will mostly be lesser than the time taken to process an equivalent number of chunks from $I_4$. This leads to a non-strict linear scaling.

5.4 Summary and Discussion

This chapter investigated the role of accuracy-related trade-off mechanisms in improving end-to-end performance of large multi-dimensional data analysis workflow applications when scalability alone does not suffice. Users must be willing to sacrifice the quality of output for high-level analysis tasks to avail of these trade-off mechanisms. An important aspect to trading analysis accuracy is the need to support different kinds of application-level quality-of-service (QoS) requirements put forth by the user. Using an example borrowed from a high-level analysis workflow for

187
classification of large image data, we show how simulation studies can model the relationships between accuracy parameters and metrics of interest for a given set of user requirements. The knowledge gained from these simulations are then incorporated within heuristics that can dynamically reconfigure the analysis tasks during execution. Parallel implementations of these heuristics on cluster systems were shown to provide improved response to application-level QoS requirements under different sets of experimental conditions. The accuracy trade-off configurations are implemented as extensions to the application-centric middleware designed to run parallel dataflow programs (Figure 5.12).

Figure 5.12: Summary of contributions in this chapter (highlighted in red)

Our important contributions and observations from the experimental evaluations are:
- An important accuracy trade-off mechanism in the analysis of large high-resolution multi-dimensional datasets is the coarse- and fine-grain sampling or approximation of the regions of the dataset. We developed heuristics that can identify and select the more favorable data entities for analysis towards satisfying application-level QoS requirements under time constraints.

- Object-based heuristics rely on the predictable accuracy and performance behavior of individual data entities, while region-based heuristics are applicable when the relationship between accuracy and performance cannot be estimated; in such cases, the spatial locality of feature content can be used to get reasonably good estimates.

- Parallel implementations of these heuristics on cluster systems showed good improvements over execution strategies that did not configure any accuracy parameters.

- The differences in improvement from one image dataset to another in our results reiterated the fact that accuracy of analysis is dependent on the data content.
CHAPTER 6

CONFIGURABILITY REQUIREMENTS AND HIGH-LEVEL SPECIFICATION

This chapter lays the groundwork for the high-level specification of different aspects of an application’s execution context that contribute towards its end-to-end performance. At an abstract level, we can consider the various analysis requirements of tasks in our applications as motifs that typify the domain of multi-dimensional data analysis. Chapters 3, 4 and 5 described techniques to address the performance demands of scientific multi-dimensional data analysis applications. It can be observed that performance optimizations brought about by these techniques at large data scales require some form of parameter configuration within one or more system middleware layers. Each configuration is closely related to the functional properties that characterize specific motifs. For example, if an analysis task comprises a set of operations where accuracy trade-off mechanisms may be carried out at the granularity of data objects, then the task could be configured using our object-based heuristics (Section 5.3.1) to improve system response to application-level QoS requirements.

As the size of datasets and complexity of analysis applications continues to grow, there is a multi-parameter expansion in the configuration space for the applications. User-directed approaches to configuration within this space are infeasible. Expecting
end-users to provide precise analytical models for the system to predict the performance of each analysis task in the workflow for configuration purposes is also not realistic. System-directed approaches or autonomous approaches typically combine available information about the application execution context with heuristics or past history of execution in an attempt to determine performance-optimal configurations.

These approaches are usually based on application-agnostic structural and runtime aspects of applications such as number of tasks and the amount of data exchanged between tasks. These approaches also assume the functional equivalence of tasks, global applicability of parameter values, and independence of data content as a result of which they are often not universally applicable to entire workflow applications, but rather to low-level sub-workflows where the corresponding assumptions hold.

Our work in the previous chapters demonstrated the importance of incorporating domain-specific optimizations for requirements within a domain. For existing legacy applications that require the analysis of large multi-dimensional data, we showed that a careful coordination of data retrieval, partitioning of datasets, distribution of data chunks among nodes, and orchestration of computation-communication overlap in parallel across nodes is highly beneficial and necessary from a performance perspective. A special emphasis was laid on dynamic reconfiguration of high-level analysis tasks as an accuracy trade-off mechanism to meet performance and QoS requirements. The middleware tools and system-directed configuration approaches available to the scientists are not flexible enough to support such coordinated performance optimizations out-of-the-box for application instances within specific domains. The ramifications of this limitation are two-fold and affect different classes of users:
• If scientists encounter challenging performance requirements posed by new application instances within the multi-dimensional data analysis domain, and these challenges have a similar flavor to the ones investigated in this thesis for legacy applications, the scientists need easy access to relevant existing configuration services to improve end-to-end performance at large data scales without having to re-design and implement low-level configuration services from scratch. Even if the scientists possess any domain knowledge about the functional properties of the analysis operations that can point the system to the usage of appropriate configuration services, such knowledge cannot be currently expressed to the system without considerable programming effort.

• If the developers of codes for applications within a domain discover new functional properties of analysis operations (different from the ones described so far in this thesis), then they need to express the accuracy and performance behavioral characteristics associated with various parameter configurations to the system. In the absence of any integration across middleware layers, scientists have to design and implement low-level configuration services specific to every system architecture and have application instances tightly coupled with the configuration services.

**Domain-specific Configuration-as-a-Service.** We motivate and describe mechanisms to improve end-to-end application performance by allowing the *specification* of a set of high-level directives and knowledge to assist systems with the *configuration* of low-level execution plans to meet specific application requirements on specific systems. In this sense, our approach advocates taking the middle ground between purely
user-directed and system-directed approaches to support analysis requirements. For our target domain of applications which analyze high-resolution multi-dimensional datasets at very large scales, my extensive experience gained through understanding of multiple application instances, as demonstrated in the previous three chapters, has provided a conceptual knowledge set that I used effectively to develop configuration services at different system layers to achieve various degrees of performance optimizations. This conceptual knowledge could include the functional properties of analysis operations, structural properties of workflow tasks, their interplay with system architectures, characteristics of the datasets, and the end-user requirements. Now, for new application instances to benefit from configuration services, we need mechanisms to express an application’s configurability using high-level directives.

Our motivations are similar to those of high-level directives and data distribution constructs supported within a data-parallel framework like High Performance Fortran (HPF) [57]. In HPF, the high-level constructs are extensions to the F90 array language to allow users to control data partitioning and mapping across multiple processors, and also assist the HPF compilers in translating this high-level user specification into an executable program by generating the low-level communication code for a given set of data-parallel operations and data distribution directives [85]. HPF compilation and execution follows a two-phase process. The actual distribution of data and computation is not explicitly specified in HPF, but is left to the compilers to generate execution plans with the help of the high-level directives. Through directives such as block and cyclic data distribution across processors, and constructs to support parallel computations within an iteration space, HPF programmers can specify how
array data must be laid out, and how array computations must be carried out at a high-level.

Figure 6.1: Specification and Configuration within an application instance

Figure 6.1 depicts our proposed approach to support domain-specific configuration of applications for performance optimizations as a three phase process. The first phase allows for high-level specification of conceptual knowledge within an application execution context. Unlike HPF, this knowledge is not limited to data distribution and parallel computation constructs alone. Portions of the knowledge may come from multiple sources, and the knowledge from one source may conceptually be related to
that from other sources. We need a unified, homogeneous model to represent this
global knowledge for an application at a high-level so that the different sources can relate to each other. The second phase is a planning or transformation phase that translates the high-level analysis requirements in the specification onto appropriate conceptual service configurations. In the execution phase that follows, the service configurations developed in the second phase are realized via low-level execution plans optimized for a specific system architecture.

This chapter focuses on the high-level specification part of the above process. My work addresses two important research aspects in connection with high-level specification, namely: (i) What information about large multi-dimensional data analysis constitutes useful conceptual knowledge for applications within our target domain? In other words, what kind of high-level information can users of such applications expect to provide? To this end, I formally investigated the configurability requirements of analysis tasks in our motivating applications that potentially lead to performance optimizations, and determined a set of high-level, inter-related concepts that can be used to capture these requirements concisely. Section 6.1 summarizes the configurability requirements of multi-dimensional data analysis applications, and the briefly describes how the parameters can be configured to enable translation of high-level specification of user queries to low-level execution plans. (ii) When all useful conceptual knowledge for an application instance have been determined, how do we represent the knowledge in an efficient manner? From a specification point of view, users must be able to easily understand and express their knowledge at a high level. From a configuration point of view, all knowledge must be expressed in a form

\(^{38}\)To enable the automated translation of analysis requirements onto appropriately configured services, the conceptual knowledge must be expressed in some machine-comprehensible form.
that is machine-comprehensible to allow for automated translation from specification to execution plans. To this end, I investigated the use of application description techniques borrowed from the semantic web world, including ontologies and rulesets. Section 6.2 describes my contributions towards capturing conceptual knowledge for an application workflow using well-defined semantics.

6.1 Application Configurability Requirements

Based on our prior experiences with multi-dimensional data analysis applications in our target domain, we have identified a set of parameters that can influence an application’s end-to-end performance. These parameters are configuration decisions or knobs that can be tuned in order to meet specific user requirements on different components of an application (e.g., nodes and links of a workflow). In this section, we describe the configurability specification requirements of these applications – namely, what performance parameters must be specified at a high-level for configuration purposes? We categorize these requirements into: component-level, workflow-level, and domain-specific accuracy trade-off requirements.

6.1.1 Component-level Configuration

Components or ensembles are building blocks of application workflows that are bound to actual pieces of code during execution. Here, we highlight key requirements to enable performance-optimal configuration of components in our target applications.

Data Partitioning: Strategies used for partitioning large multi-dimensional datasets play a significant role in the performance as the component instance needs to be suitably configured to process the partitioned data. For example, in confocal microscopy
images processed in the PIQ workflow, the image data are characterized by very high resolution data along the X- and Y- spatial dimensions but low resolution along the Z-dimension. For maximum utilization of computing resources and for performance reasons, we need a strategy that partitions (each slice of) the dataset along X- and Y-dimensions so that we have a sizeable number of partitioned data entities or *chunks*. Similarly, for each data point along the Z dimension, dense data could be partitioned uniformly, while strategies like recursive bisection could be employed when the data has a non-uniform distribution, for load balancing and eliminating straggler nodes.

Users may possess different levels of information that could be used to assist the configuration of components:

1. The end-user (workflow composer), based solely on the knowledge of the dataset instances involved, may explicitly specify a partitioning strategy:

   (a) Partition dataset along dimensions \(d_1\) and \(d_2\) into 2D chunks of size 100 × 100 data points each.

   (b) Employ a uniform 3D partitioning strategy for dataset along dimensions \(d_1\), \(d_2\) and \(d_3\). Retain the same partitioning across all other dimensions.

   (c) Partition dataset along dimensions \(d_1\) and \(d_2\). Use recursive bisection to partition each slice independently.

2. Explicit information about the partitioning strategy may not be available with any one given user. In such cases, partial information from different sources may need to be used to derive a suitable strategy:

   (a) The end-user may only possess metadata properties of the dataset instances: e.g., the resolution of data points along dimensions \(d_1\), \(d_2\), and \(d_3\)
are in the ratio of 1000:1000:1, or, the relative density of data points across regions remains the same across the Z dimension, and so on.

(b) The component developer may not possess \textit{a priori} knowledge of specific data instances that the component will operate on. But this class of users may provide information regarding the execution conditions that the component is optimized for: e.g., this component works best with \( n \)-dimensional data chunks where \( n \leq 3 \), or, this component gives best performance when the number of chunks is more than twice the number of available processing units, and so on.

(c) Knowledge within the spatial decomposition domain (that neither class of users may provide) may suggest for example, that quad-tree based schemes provide optimal partitioning when the density of data points within different regions of a dataset follows a certain distribution.

\textbf{Data-parallel and Heterogeneous Components:} A large number of analysis operations avail of data parallelism for improved performance. Different portions of a single large data entity, or a large number of smaller data entities altogether can be processed concurrently. In other words, the execution of such components can be viewed as a bag-of-tasks running on a set of computing resources, where each task is an independent instance of the component operating on different data. On the other hand, a number of components and ensembles in our target application domain are not inherently data-parallel in nature. Such components are characterized by the need for large amounts of disk I/O and data communication between processes during
execution. The filter-stream programming model has been demonstrated to handle such components efficiently. Regardless of the nature of the components, we require mechanisms to specify high-level configuration directives to assist in the mapping of bag-of-tasks or filter instances onto the compute nodes.

**Summary of Component Configuration Requirements:** The requirements for component (or ensemble) configuration to improve their performance can be summarized as follows:

1. The first set of requirements deals exclusively with the data regardless of the components and compute resources. There is a need for rich metadata descriptions of the multi-dimensional datasets within the domain and of the various relationships involving the spatial dimensions of these datasets. This would allow the expression of explicit notions such as dimensionality of the datasets involved, dimensions along which to partition data into chunks, the dimensions of the chunks, and also implicit domain-specific relationships such as spatial locality between chunks or dimensions among other useful configuration-related information.

2. The second set of requirements deals with component descriptions and data distribution but is independent of the underlying system. We need to have lightweight built-in configuration primitives corresponding to well-known spatial decompositions. The user must be able to specify chunk-based partitioning and mapping schemes at a high-level and the corresponding primitive, fused with the original component or ensemble, must take care of setting up the bag-of-tasks or filter layout as the case may be. This configuration primitive setup
will functionally differ depending on the nature of the component. For data-parallel components, there is a one-to-one mapping between data instances and tasks. That is, a new task is created for every data instance, and is added to the bag of tasks. A data instance could correspond to a single data chunk, or a collection of chunks (such as a 2D slice or a 3D stack) depending on the dataset description. During execution, each task is treated independently and invoked on a compute node within the pool of resources. For heterogeneous components, a fixed number of filter instances are created to begin with. This number may be chosen by the user or may be set dynamically based on the layout specified by the user (e.g., place two instances of filter A on each compute node within the resource pool). Here, there is a many-to-one mapping between data instances and filter instances. That is, one or more data instances may be mapped to each filter instance during execution. In addition to high-level specification of the layout dataflow (e.g., one-to-one or all-to-all stream connectivity between filter instances), we also need primitives that can map data instances to filter-instances based on well-known distribution strategies. Examples include static strategies such as round-robin, stacked or space-filling curve-based mapping, and dynamic strategies such as on-demand mapping enforced by the system at runtime.

3. The third set of requirements deals with component descriptions and knowledge that assist in job scheduling, i.e., the mapping of tasks and filters to the compute nodes at runtime. If bag-of-task style execution is being performed on heterogeneous tasks (e.g., when each task processes a data chunk with different data volume or density), we need mechanisms to annotate the coarse-grain
component with such descriptions about its tasks. These descriptions could be combined with knowledge from the domain of job scheduling so that appropriate heuristics with backfilling extensions may be used to improve the overall execution time. Similarly, for filter-stream style execution, the mapping of filter instances to compute nodes can be configured based on such descriptions and user requirements. If the user’s preference is to minimize the data communication overheads, or if the network bandwidth is a perceived performance bottleneck, then filters that operate on the same set of data instances can be co-located on the same set of resources. On memory-constrained resources, runtime systems must be configured to stage intermediate data to local disks via specialized mediator filters should the need arise. On multicore nodes, transparent copies of filter instances can be spawned at runtime to minimize computation time. It would be useful to provide the runtime system with hints concerning the number of transparent copies for each filter instance mapped to a node.

6.1.2 Workflow-level Configuration

Components or ensembles and links between them constitute the structural entities of any given application workflow. In this section, we discuss some of the configuration requirements that involve more than just a single component or ensemble, i.e., where optimization decisions involve multiple components and links of the workflow and the relation between them. All these requirements build on those specified in section 6.1.1, namely, the need for mechanisms to specify rich metadata descriptions of multi-dimensional datasets.
Data Transfer Between Components: When multi-dimensional data is processed and exchanged between workflow components or ensembles, the dimensions, shapes and sizes of data entities are often preserved throughout the workflow. That is, a single data partitioning strategy is usually chosen across all components in the workflow. However, in certain cases, the optimal partitioning strategy for one set of components may differ from that of another set of components within the same workflow instance. For extremely large data, repartitioning entire datasets to ensure ‘locally’ optimal partitioning for each component may not be feasible. Instead, globally optimal partitioning strategies which may not necessarily ensure locally optimal partitioning for all components are typically used where the number of repartitionings are minimized.

In this section, we present a set of requirements that deal with improving performance through on-the-fly manipulation of data as it traverses from component to component in a workflow instance. These requirements are independent of the underlying system to which the workflow is mapped.

1. In cases where the same dataset partitioning strategy is employed across the entire workflow, and data exchange is achieved via disk-resident files, performance for certain components can be improved by reorganizing the layout of the data within chunks (intra-chunk reorganization). Since chunks by definition can fit within physical memory, this form of reorganization could potentially lead to performance improvements without adding substantial overheads. Within each chunk, multi-dimensional data could be organized in different ways based on the spatial dimensions – the popular ones being row-major and column-major.
With a fixed spatial ordering the other dimensions could again be organized in several ways; For example, striped organization is one where all non-spatial attribute values of a data point are stored together, followed by those of the next data point (as dictated by the spatial ordering), and so on. Another example is stacked organization where the data is laid out by dimensions, i.e., for a given attribute, the values of all data points are stacked over those of another attribute, and so on. Now, depending on the characteristics of the analysis components, the appropriate organization can be chosen for performance benefits. For example, if it is known that a given component instance will access only dimension Red from among the three color channel dimensions, then having a stacked organization of data within a chunk would minimize the number of seeks, thereby reducing the disk I/O overheads. Therefore, the first requirement for workflow-level configuration of data transfer is the ability to specify how data within a chunk is laid out, and how components seek to access data within chunks. We also need configuration primitives to perform different forms of intra-chunk reorganizations. Users or the system should be able to place these primitives at one or more locations along the path between two components. During execution, the actual reorganization may be performed by the hardware or software, or potentially even by the links using network processors.

2. When the same data partitioning strategy is employed across all workflow components, performance for certain components could also be potentially improved by choosing an appropriate layout for entire chunks within a file (inter-chunk reorganization). Here again, the goal is to improve performance by minimizing seek overheads and read latency. For example, when a component A in the
workflow produces a 3D collection of chunks as output, the layout of this output data on disk, i.e., the mapping from chunks to on-disk files is done so as to optimize the writes. However, a downstream component B in the workflow may attempt to read chunks in a manner that is sub-optimal given the chunk layout of its input data. Hence, the second requirement for workflow-level configuration is the ability to specify mappings between chunks and on-disk files – does the system need to maintain a large number of small files (with few chunks per file) or a small number of large files (with many chunks per file); in the latter case, does a component prefer if chunks are laid out in row-major order or in column-major order within the files? Similar to the previous requirement, we need configuration primitives to perform inter-chunk reorganizations within files, along with the ability to fuse these primitives to one or more components in the workflow.

3. Many multi-dimensional data analysis applications involve aggregation-style operations, where individual components need to compute global data structures (such as prefix sums or histograms) over attribute values of all points within a dataset. These data structures may be computed for a single dimension or for all dimensions. They are then used to speed up the execution of one or more downstream components. For large datasets, this computation may incur significant communication overheads when performed repeatedly over many dimensions. Substantial performance improvements can be achieved by having such components produce only partial data structures (by way of processing done only on local data – individual chunks or within other building-block entities that
constitute the dataset), as this removes the need for large data communication. However, the downstream component now needs a mechanism to generate global aggregates for individual data points on the fly using the partial aggregates. Typically, the benefits of partial computation outweigh the overheads introduced into the downstream components leading to improved end-to-end workflow performance. Hence, the third requirement for workflow-level configuration is the ability to describe if components are capable of generating partial output, and also the nature of the partial output such that other components can use them if possible. Based on the nature of the partial output (e.g., along which dimensions have partial aggregate been computed? what threshold values were used for each dimension?), we need configuration primitives that can be fused with downstream components that assist in the computation of global values from the partial output.

**Mapping Workflows to Compute Nodes:** The first step in the execution of workflows with multiple components is workflow scheduling: the high-level mapping of the coarse-grain components and ensembles onto the compute nodes within a resource pool. The set of requirements in this section correspond to different workflow mapping scenarios and deal with strategies for performance improvements.

1. In the first scenario, the end-user (workflow composer) may explicitly specify the mapping between workflow components and compute nodes. On account of different reasons such as data security that are unrelated to end-to-end performance, the user may wish to bind each workflow component to specific sets of compute nodes. This binding may be expressed on a per-component basis. During execution, the appropriate executable is invoked for component
instances bound to each site. A minor extension of this scenario is one where
users may pool components and ensembles into logical groups, and map groups
onto common sets of resources instead. An equivalent version of this scenario
is where users select regions of workflows (and not individual components) and
map the regions directly onto pre-specified resource pools. In the latter sce-
narios, appropriate strategies can be applied to enable optimal sharing of the
set of nodes from the resource pool by the logical groups and regions that have
been mapped to it. The different regions of the workflow could be treated as
coarse-grain bags of tasks or they could bear some data dependencies. The
strategies and requirements in this scenario mirror those specified in item 3 of
section 6.1.1.

2. The second scenario is related to the first in that the end-user may explicitly
specify only a partial set of component mapping constraints. In the worst case,
the end-user may not specify mappings for any of the workflow components.
In such cases, alternate sources of information must be harnessed to provide a
performance-optimal mapping:

(a) The component developers may possess some information about functional
aspects of their components that is independent of actual data instances.
Relative performance models obtained through profiling could then be used
to guide the workflow scheduling: Component $A$ spends 80% of its time
doing computation, or, the volume of output produced by ensemble $B$ is
roughly a third of its input data volume.
(b) Knowledge within the domain of spatial data analysis operations may recognize any component instance that spends 75\% or greater of its overall execution time per chunk purely on computation as being a compute-intensive component.

(c) Knowledge within the domain of workflow scheduling may recommend that compute-intensive components be preferably executed on faster nodes, while those components classified as communication-intensive to execute on nodes equipped with high-speed interconnects.

As can be seen here, when workflow mapping is not explicitly specified by the end-user, partial information possessed by different sources can be harnessed to provide hints or directives to the system.

3. The final scenario is one where neither class of users provide any information about the workflow and its components. In this case, the mapping needs to be performed entirely by the system based on whatever knowledge may be available within different domains (such as multi-dimensional data domain) and prior history of execution. During execution, application-agnostic data-intensive workflow scheduling techniques need to be applied to map the workflow to the compute nodes.

### 6.1.3 Accuracy Trade-off Configuration

A special emphasis of this thesis is on supporting different kinds of accuracy performance trade-offs in large-scale, multi-dimensional data analysis. In this section, we present a set of requirements for the configuration of application workflows stemming from the need to support such trade-offs.
6.1.3.1 Extended Workflow Model for Adaptness

The basic workflow model is comprised of abstract analysis components and links between them along which data is exchanged. Each component or ensemble is described in terms of the number and the types of input data it can operate on, and in terms of the output data types that it can produce. Some models also capture the structural properties of well-identified workflow analysis patterns. In certain cases, the model also includes mechanisms to propagate metadata descriptions along components and ensembles so that properties of output data can be associated with those of the input data. For adaptive applications and support for application-level quality-of-service, this basic workflow model can be extended through the use of constraints, requirements, quality metrics and accuracy trade-offs.

1. **Constraints** include any limitations on the availability of resources such as time, number of computational cycles, energy consumption or bandwidth utilization which necessitate the exploitation of adaptivity. For our target domain, we focus our attention on time-constrained application workflow instances. Users may specify that the workflow execution for a dataset instance must be completed within the available time $t$. Users may also impose fine-grain constraints on the execution time of individual components or workflow regions such as stating that selected components must execute in real-time, or that certain workflow regions must produce output within time $t$ of receiving their input.

2. **Quality Metrics** are specified by users as a means to represent the accuracy of the analysis. The quality metrics could be associated with individual components, or with workflow regions, or with entire workflows. These metrics could
be generated by analysis components along with their output, or could be computed separately based upon the metadata properties of the output data. In confidence-based methods, the confidence value can be used to represent the accuracy of classification. On the other hand, operations like image warping may not have any automatically generated values to represent the accuracy. In such cases, lightweight statistical measures can be obtained by comparing the obtained result against some ground truth, and the accuracy of the analysis can be computed based on these similarity measures. The quality metric may capture not only how accurate the analysis is, but also how much of an error was introduced.

3. **Requirements** or domain-specific application-level quality-of-service (or aQoS) requirements are similar to constraints with a couple of exceptions: (i) While constraints are associated with resources, aQoS requirements are associated with quality metrics, and (ii) With constraints, the goal is often implicit (for e.g., minimize time, minimize energy consumption, maximize utilization of parallel processors, etc.). However, with aQoS requirements, the goal depends on the domain-specific quality metric and needs to be explicitly specified by the user. (for e.g., users may wish to maximize a metric such as confidence, but minimize a metric such as classification error).

As a result of the above extensions, in addition to the basic workflow specification, the user can also provide high-level constraints and requirements. A simple example is “Given the Neuroblastoma Classification workflow (NB), maximize the average confidence in classification across all data regions within 15 minutes; Each chunk must be classified with a minimum confidence of 90%”. Here, NB is assumed to
be represented using a basic workflow model where the output of the workflow is a set of classes corresponding to the regions of the input dataset, the aQoS goal is to maximize the confidence in classification across all data regions, the constraint is a time limit of 15 minutes, and the user-specified bound on the accuracy is captured using the aQoS constraint that each classification should be done with at least 90% confidence. Similarly, the users could specify constraints, goals and requirements associated with individual components or regions of the workflow. To meet such constraints and requirements, the extended workflow model must exploit the adaptive nature of applications by supporting one or more accuracy trade-offs as discussed in the following section.

6.1.3.2 aQoS Provisioning Strategies

We have identified a set of strategies that can be (and have typically been) employed within our target application domain to meet user specifications on aQoS. These strategies can be classified into (i) those that preserve the dataflow, and (ii) those where the dataflow can be manipulated so long as the desired data products are obtained. Strategies that do not preserve dataflow can further be classified into (iia) those where original workflow structure is unaffected, and (iib) those where the original workflow structure can be modified. In the remainder of this section, we provide an elaboration of these strategies.

**Dataflow Preserving strategies** In this set of strategies, the original flow of data between components, as specified by the user in the workflow description, is preserved during execution. That is, beginning with the input datasets, workflow components
process all their incoming data and produce their respective output data. The accuracy of analysis and performance levels of individual components and ensembles are tuned based upon the extent to which input data is processed.

When all the input data to a component must be processed, the constraints on time and aQoS requirements can be met by configuring components to do one or more of the following:

1. If the component instance requires multiple algorithms being to be applied to the input data instance (e.g., the application of several models to classify input feature sets into different categories), the accuracy of analysis of the component instance is computed by aggregating or taking a consensus view across the results obtained from all the algorithms (e.g., in the NB workflow, eight different classification algorithms are concurrently used to classify each data chunk based on its features. The confidence in classification for each chunk is computed based on a majority of votes made by each algorithm). When faced of time constraints, instead of executing multiple algorithms, one can reduce execution time by applying only a fraction of the algorithms. All input data is processed, however, the quality of the result is affected because all algorithms or models are not considered. The user can control the quality through aQoS requirements that place limits on the minimum number $n$ of algorithms that must be applied, or on a minimum accuracy threshold that must be achieved.

2. From the point of view of a parallel component composed of many component instances (tasks or filters) processing different data instances, the quality of the result for the component as a whole is computed using the results from individual component instances. Depending on the feature content within each data
chunk, the corresponding component instance may use different combinations of algorithms to process that chunk.

3. If there is only a single algorithm corresponding to each component instance, then certain characteristics of the algorithm can be exploited to trade accuracy for performance. In our strategies, we wish to be minimally intrusive, i.e., we treat the component instances as grey boxes to which appropriate parameters can be passed during execution. A well-known strategy for algorithms that exhibit an iterative computational structure is loop termination, where the execution can be curtailed prematurely by specifying a limit on the maximum number of iterations; the result obtained at this point would generally be less accurate than that obtained using all iterations. Similarly, models constructed using large multi-dimensional datasets, may often utilize polynomial functions of some degree \( n \). However, under time constraints, one can place a bound on the degree of the polynomial function used to some value less than \( n \), such that execution time is improved at the cost of the accuracy of the model. Importantly, all the input data is still processed by the component instance.

4. Specific to this domain, the data models or data representation techniques employed during the computation can also be used as a handle to control the accuracy and execution time. For example, if a component instance must determine the structural similarity between two or more spatial data entities with complex shapes, shape approximation techniques (such as minimum-bounding polygons/spheres or shapelets) can be employed to reduce the amount of computation required. For certain types of operations, the component instance may
also resort to transforming input data into a different domain (e.g., wavelet transformations) and performing appropriate computations in the transformed domain to obtain faster results, albeit of different quality.

**Non Dataflow-preserving strategies** In this set of strategies, the intended flow of data between components, as specified by the user in the workflow description, can be modified during execution to meet the constraints on time. Data can be filtered or processed as it traverses links between components such that the components may not receive all the input data that they would otherwise have. The analysis operations within the components themselves remain largely unaffected; hence, the accuracy of analysis is affected by the fact that the components operate on reduced input data.

Examples of commonly employed strategies under this category include:

1. Fine-grain reduction comprises of a set of strategies to reduce the amount of data within individual data instances that are processed by a component instance. For example, assume that each task from a bag of tasks corresponding to a time-constrained data-parallel component operates on a single data chunk. Now instead of feeding each task entire input chunks, we can subject the chunks to data reduction strategies such as multi-dimensional data sampling, resolution reduction, or tessellation so that each task operates on lesser data amounts. Processing lesser data generally reduces execution time, however it need not necessarily decrease the accuracy of analysis for the task. For the chosen data reduction strategy, the appropriate technique (e.g., uniform or biased sampling, summation or average-based tessellation) can be used depending on the application execution context, and different strategies could be chosen for different tasks. If a component processes feature sets, dimensionality reduction schemes
can be used to reduce the input data volume. If a component instance processes multiple chunks (e.g., a projection operation operating on a stack of chunks), then, one could similarly employ strategies to reduce the volume of the input stack for each task.

2. Medium-grain reduction is typically applied at the level of entire components or ensembles as opposed to individual tasks or filters. Under time constraints, one can select a subset of chunks from the input dataset to process. This subset can be selected statically or dynamically during execution. When done dynamically, the problem reduces to determining an order of execution of the chunks in order to best meet the aQoS goals. Different strategies could be used to determine the order of execution including sampling-based, region-growing and space filling curve based approaches depending on the characteristics of the data and the execution context.

3. Coarse-grain reduction is applied at the workflow level and could involve multiple input data instances to a component or multiple components. Suppose a component processes two input data instances such that for every chunk from the first instance, it processes \( n \) chunks from the second instance. Now, adjusting this ratio such that the component now processes only \( k \) chunks (where \( k < n \)) of the second instance is an example of such a strategy to achieve accuracy trade-offs. Such a scenario occurs when the analysis involves two or more data instances that are partitioned using different strategies into chunks of various dimensions. A final strategy is one where entire links or components
within a workflow can be dropped so long as the validity and consistency of the dataflow is not affected.

As shown in this section, trading accuracy for performance can be accomplished through many strategies. With each strategy comes a set of configuration requirements to ensure that performance is optimized and the aQoS goals are met. An emphasis of this work is on configuration mechanisms that provide a best-effort execution in response to the aQoS requirements. That is, the configuration seeks to provide the best possible system response to the aQoS requirements within the constraints of time. The configuration of the application workflow itself may include the simultaneous use of one or more of the above strategies. Some of the strategies described for components above, may equally be applicable to ensembles and to regions of workflows.

6.2 High-level Application Specification

**Background.** A key issue in supporting the synthesis of data intensive computation and data management is to liberate users from low-level programming details, by describing applications functionally and in a manner that is independent of the underlying infrastructure. The availability of high levels of abstraction for describing data analysis is important regardless of the computing model employed or the underlying architecture. Traditionally, in relational database systems, users express the data analysis operations in the form of a declarative querying language such as SQL assisted by procedural language extensions to provide control flow such as loops. SQL is an abstraction that presents users with high level data concepts like tables, rows and indices, as well as high level operations to select data, insert or delete data, join
data from multiple tables etc. In large-scale parallel data processing systems such as Dryad [55] and MapReduce [38], the focus is more on larger data scales and simpler fault tolerance and consistency mechanisms than database systems. Here again, users are provided a set of high-level primitives to express the partitioning of the data and applying functions to the data partitions. Currently, an area of very active research, there are many efforts including Sawzall [90], Pig Latin [88 87], JAQL, DryadLINQ [114] and Hive, that are seeking to develop high-level data flow languages atop data processing systems such as MapReduce in order to ease program development. Each of these tries to provide stylistically different programming approaches – while Pig appears to extend SQL, Sawzall syntax is heavily influenced by Java and Pascal. A consequence of describing application structure with greater transparency of execution is that the onus of performance optimizations for the applications falls on the system.

An analogous trend has also been observed in the world of scientific big data applications, where workflow management systems are popularly used in the Grid-based execution of complex data analysis application pipelines. Workflow-based applications are already naturally broken down explicitly into a set of components (with dataflow dependencies). Workflow languages enable users to describe workflow-based applications at a high level of abstraction. Users need only to functionally describe the components and types of data that are exchanged between pairs of components. Workflow technology was first developed by the business community to model business processes. The Business Process Execution Language (BPEL) [39] is an XML-based modelling language used for describing business processes and flow control, and has

also been adopted as the de-facto standard for orchestrating workflows from Web services. Scientific workflows, on the other hand, differ from business workflows, in that scientists need tools to string together data analysis components for execution on distributed resources, and exercise control over the data flow between the components. The scale of data and resources involved in scientific workflows calls for more robust, dependable systems than those used for business workflow management.

There is a large body of scientific workflow systems in use today, the notable ones among them being Kepler [79], Pegasus [39], Taverna [86] and Triana [103]. [113] and [19] provide a comprehensive survey of these systems, and classify them based on their functionality and applicability. Workflow languages in these systems provide users with different means of describing their workflows. Taverna, a services-based, Grid-aware workflow system uses a proprietary language called Scufl (Simple Conceptual Unified Flow Language), a high-level data-centric XML-based language to allow users to define a workflow using Web and Grid services connected with data links. Triana supports multiple workflow languages including Web Services Flow Language (WSFL) and BPEL for describing services-based workflows. These systems are used when applications are cast as workflows of services in distributed environments. Workflows within Kepler are serialized in an XML-based language called Modeling Markup Language. Users express workflow components as actors that interact with each other through channels. Pegasus is slightly different from the others in that users do not compose workflows from existing services. Instead, the workflows are composed of jobs that perform computations on any resources in a distributed environment. These jobs take in data from massive datasets and produce
data products. With Pegasus, users can focus purely on workflow design and describe abstract workflows that can be mapped to any distributed set of resources on the Grid.

6.2.1 Semantic Representations for Workflow Applications

Semantic representations for application workflows describe exactly what the role of every part (component and link) of a workflow is from the point of view of a scientist or a domain expert. Semantic descriptions are achieved through the integrated use of ontologies and semantic web-based languages, such as the Web Ontology Language (OWL)\(^{40}\). The main ingredients of such ontology languages are concepts and properties, where concepts are abstract entities and properties help establish relationships between concepts or instances of concepts. In general, semantic descriptions are better than mere syntactic descriptions because they allow for effective reasoning (i.e., inferencing) when expressed in machine-processable form. A workflow system that uses full ontological integration for application description has several advantages including cross-checking between ontologies, addition of new concepts into the system and discovering new knowledge (via inferencing) within the system.

The early motivations for using semantic representations in workflows were (1) to allow users to discover or query for components or services from a library or repository of components for the domain, based on their descriptions, and (2) to support workflow validation; When workflows with hundreds of tasks need to be created for large datasets, semantic descriptions can assist users in creating valid workflows, by performing type and consistency checking for data exchanged between components. However, semantic descriptions are being increasingly used for other purposes aimed at making things easier for the user. By using ontologies to denote

\(^{40}\)http://www.w3.org/TR/owl-ref
relationships among concepts in the geospatial domain, datasets and services, Chiu et al. [34] show that all candidate workflows that can service a high-level user query within that domain can be automatically composed. Nadarajan et al. [81] and Qin [91] show how case-based reasoning and ontologies can be used for automatic workflow composition within specific application domains. With this support, users need not compose workflows from scratch, but instead can choose from one of the automatically composed workflows that best suits their needs.

The Pegasus system uses Wings (Workflow Instance Generation and Selection) [46] to provide semantic descriptions for abstract workflows that are composed either manually or automatically. Wings uses semantically rich representations of workflows to express repetitive computational structures in a compact manner and to describe underspecified data collections [46]. Datasets have metadata that describe their size, cardinality (number of data elements) and types of each data element. In Wings, semantic representations are used for the data, where datasets and data elements are concepts within a core data ontology, and the metadata are the properties. These representations serve three purposes: to describe relationships between different data elements, for metadata propagation and for provenance-based querying. Wings also uses semantic representations for components in a workflow to describe their functional properties, their input and output data types, and the relationship between their input and output data, the constraints on the type and values of their input arguments. By describing the relationship between the input data elements and the intermediate and final data products, Wings can automatically propagate the metadata properties along the workflow components from the input data to the data products.
6.2.2 Specification of Multi-dimensional Datasets

One of the important constituents of the conceptual knowledge for a given application instance is the metadata properties of the datasets within the application domain. Semantically rich metadata descriptions of large multi-dimensional datasets need to capture the dimensions and shapes of individual data entities in the datasets, the distribution of data along each dimension, the relation between data along one or more dimensions and the spatial locality properties at different regions in the dataset. Systems like Wings provide “core” ontologies that can describe abstract components and data types. For any new application domain, these core ontologies can be extended to capture domain-specific information. Semantic descriptions for new data types and components are maintained in domain-specific ontologies. Workflow templates for applications within that domain can then be constructed using these ontologies. The core data ontology in Wings contains OWL-based descriptions of generic data concepts such as File, Collection of Files of the same type, and CollOfCollections, as well as their metadata properties. The core data ontology cannot capture the semantics of datasets used in our motivating applications in a well-defined manner. So, we extended this core data ontology (as shown in Figure 6.2) to express concepts and properties in the spatial, multi-dimensional data model so that application data flow can be conveniently described in terms of these concepts. While this ontology is not exhaustive, it is generic enough to represent typical application instances such as the PIQ and Neuroblastoma Classification workflows. It can also be extended to support a wider range of data analysis applications within the domain.
Figure 6.2: Wings data ontology extensions for spatial, multidimensional data
6.3 Summary

This chapter consolidates the learnings from the previous three chapters in an attempt to provide a high-level specification layer for conceptual knowledge about applications in the domain of large multi-dimensional data analysis. Inspired by high-level directives and constructs in High Performance Fortran, we propose a specification layer to capture conceptual knowledge about relevant aspects of an application’s execution context. Our specific contributions to this end include:

- Identifying and classifying the configurability requirements of applications in our target domain. These requirements can be expressed using concepts and metadata properties obtained from multiple sources. Developers can describe functional properties about their components (e.g., a parallel component could be conceptually modeled using a high-level parameter \( \text{req\_machine\_count} \geq 1 \) – the underlying job scheduler could then assign a dedicated number of processors to this component for execution), while end-users can specify QoS requirements and hints for execution (e.g., if a component is an accuracy-trading component, constrain the component to use only subsampling ratios \( \geq 4 \)).

- Implementing a portion of the specification layer using ontologies and rules with well-defined semantics. The specific portion implemented was a description of datasets developed based on the spatial-multidimensional data model including concepts (such as \textit{chunk}, \textit{slice}, and \textit{stack}), and properties (metadata properties and spatial properties) that relate different concepts. The implementation was an extension to the core data ontology in a system called Wings, and was used to semantically describe the data flow in two different applications.
CHAPTER 7

IMPACT: A PROTOTYPE SYSTEM FOR APPLICATION CONFIGURATION

In this chapter, I describe and evaluate a prototype implementation of my proposed framework called IMPACT – Improving Multi-dimensional data analysis Performance via ACcuracy Trade-offs. The framework consists of three main modules. The specification module implements support for high-level specification of workflows. In this module, the application structure and data characteristics for the application domain are presented to the system. This representation is independent of actual data instances used in the application and the compute resources on which the execution is eventually carried out. The execution module is responsible for workflow execution and takes as input, the high-level description of the workflow produced by the specification module, the datasets to be analyzed and the target distributed execution environment. Lastly, the configuration module, implements techniques and runtime mechanisms to enable accuracy-performance trade-offs based on user-specified quality-of-service requirements and constraints. The architecture of the framework is illustrated in Figure 7.1. In this work, we evaluated a specific instance of this framework with well-known middleware representatives for each module.
7.1 Specification Module (SM)

The Specification Module (SM) is implemented using the Wings (Workflow Instance Generation and Selection) system [46]. In the Wings representation of a scientific workflow, the building blocks of a workflow are components and data types. Application domain-specific components are described in component libraries. A component library specifies the input and output data types of each component and how metadata properties associated with the input data types relate to those associated with the output for each component. The data types themselves are defined in a domain-specific data ontology. Wings allows users to describe an application workflow using semantic metadata properties associated with workflow components and data.
types at a high level of abstraction. This abstraction is known as a *workflow template*. The workflow template and the semantic properties of components and data types are expressed using the Web Ontology Language (OWL)\(^{41}\). A template effectively specifies the application-specific workflow components, how these components are connected to each other to form the workflow graph, and the type of data exchanged between the components. Figure 7.2(a) is a depiction of a workflow template constructed for the PIQ application.

The workflow template is data-instance independent; it specifies data types consumed and produced by the components but not particular datasets. Wings can use the semantic descriptions to automatically validate a given workflow, i.e., if two components are connected in the workflow template, Wings can check whether output data types and properties of the first component are consistent with the input data types and properties of the second component. Given a workflow template, the user can specify a data instance (e.g., an image) as input to the workflow and the input argument values to each component in the workflow. Using the metadata properties of the input datasets, Wings can automatically generate a detailed specification of the workflow tasks and data flow in the form of a DAG referred to as an expanded *workflow instance* for execution. Figure 7.2(b) shows a workflow instance generated by Wings from the PIQ workflow template.

Wings also provides compact representations in the template for component collections, i.e., cases where multiple copies of a component can be instantiated to perform the same analysis on different data input instances. The number of such copies for

\(^{41}\)http://www.w3.org/TR/owl-ref
Figure 7.2: PIQ application workflow represented using Wings

(a) Workflow Template created for PIQ

(b) Workflow Instance generated by Wings for chunksize parameter = 2560×2400 pixels
a component can be selected during the workflow instance generation stage based on
the properties of the input datasets. Performance can be improved by adjusting both
the task- and data-parallelism parameter as well as application performance param-
eters such as chunksize. For the PIQ and NB workflows, the chunksize parameter
dicts the unrolling of component collections in the workflow template into a bag
of component tasks in the workflow instance. As an example, assume that the value
of the chunksize parameter chosen for a given input image was $2560 \times 2400$ pixels
and resulted in each image slice being partitioned into 36 chunks. The corresponding
expanded workflow instance for the PIQ workflow as shown in Figure 7.2(b) con-
tains 116 tasks. Component collections are shown in the workflow instance as long
horizontal rows of tasks, signifying that each collection has been unrolled into 36
tasks, where each task represents an operation performed on a single chunk. Thus,
chunksize parameter influences the structure of the resulting workflow instance.

In our current system, we also support the notion of meta-components or explicit
component grouping. A meta-component can be viewed as a combination or cluster-
ing of components across multiple levels in a workflow template. Coalescing compo-
nents into meta-components corresponds to the adjustment of the task granularity
parameter for performance optimization. During execution, all tasks within a meta-
component are scheduled at once and mapped to the same set of resources. Note
that the use of meta-components complements the existing job clustering capabilities
provided by Grid workflow systems such as Pegasus [39]. Horizontal and vertical job
clustering capabilities in Pegasus are used to group tasks that are explicitly specified
in a workflow instance. However, it is highly desirable to also support the following
features:
• **Task parallelism:** Vertical clustering implies that workflow tasks at multiple levels are grouped into clusters so that each group can be scheduled for execution on the same processor. To improve performance, task parallelism across tasks within such groups, and also data streaming from one task to another is needed to avoid disk I/O overheads.

• **Fine-grain control:** Workflows such as PIQ also contain parallel components (e.g., MPI-style jobs) that execute across a number of nodes, but whose processes are not expressed as explicit tasks in the workflow instance. It is the responsibility of the parallel runtime system at a site to manage the processes of such components across the nodes at that site.

Meta-components can provide such task-parallelism and fine-grain control. By grouping successive components into a meta-component in the workflow template, each such component’s processes can be concurrently scheduled for execution on the same compute resource. For example, the grey rectangles shown in the workflow instance in Figure 7.3 indicate that all processes corresponding to all four components (shown as blue ovals) in the workflow have been fused into a meta-component. One task corresponding to each meta-component will be scheduled for execution concurrently. Within each meta-component task, the execution is pipelined, and data is streamed from one component to another during execution.

### 7.2 Execution Module (EM)

The *Execution Module* (EM) consists of the following subsystems that work in an integrated fashion to execute workflows in a distributed environment and on cluster systems:
Pegasus Workflow Management System is used to reliably map and execute application workflows onto diverse computing resources in the Grid [39]. Pegasus takes resource-independent workflow descriptions generated by the SM and produces concrete workflow instances with additional directives for efficient data transfers between Grid sites. Portions of workflow instances are mapped onto different sites, where each site could potentially be a heterogeneous, cluster-style computing resource. Pegasus is used to manipulate the component config parameter, i.e. the component transformation catalog can be modified to select an appropriate mapping from components to analysis codes for a given workflow instance. Pegasus also supports runtime job clustering to reduce scheduling overheads. Horizontal clustering groups together tasks at the same level of the workflow (e.g. the unrolled tasks from a component collection), while vertical clustering can group serial tasks from successive components. All tasks
within a group are scheduled for execution on the same set of resources. However, Pegasus does not currently support pipelined dataflow execution and data streaming between components. This support is provided by DataCutter [23] as explained later in this section.

**Condor** [105] is used to schedule tasks across machines. Pegasus submits tasks corresponding to a portion of a workflow in the form of a DAG to DAGMan meta-scheduler instances running locally at each Grid site. DAGMan resolves the dependencies between jobs and accordingly submits them to Condor, the underlying batch scheduler system.

**DataCutter** [23] is employed for pipelined dataflow execution of portions of a workflow mapped to a Grid site consisting of cluster-style systems. A task mapped and scheduled for execution by Condor on a set of resources may correspond to a meta-component. In that case, the execution of the meta-component is carried out by DataCutter in order to enable the combined use of task- and data-parallelism and data streaming among components of the meta-component. DataCutter uses the *filter-stream* programming model, where component execution is broken down into a set of *filters* that communicate and exchange data via a stream abstraction. For each component, the analysis logic (expressed using high-level languages) like C++, Java, Matlab and Python) is embedded into one or more filters in DataCutter. Each filter executes within a separate thread, allowing for CPU, I/O and communication overlap. Multiple copies of a filter can be created for data parallelism within a component. DataCutter performs all steps necessary to instantiate filters on the target
nodes and invokes each filter’s analysis logic. A stream denotes a unidirectional data flow from one filter (i.e., the producer) to another (i.e., the consumer). Data exchange among filters on the same node is accomplished via pointer hand-off while message passing is used for filters on different nodes. In our framework, we employ a version of DataCutter that uses MPI for communication to exploit high-speed interconnect technology on clusters that support them.

**ECO compiler**: Within the execution of a single component task, the Empirical Compilation and Optimization (ECO) compiler can be employed to achieve targeted architecture-specific performance optimizations. ECO uses *model-guided empirical optimization*[^32] to automatically tune the fine-grain computational logic (where applicable) for multiple levels of the memory hierarchy and multi-core processors on a target compute resource. The models and heuristics employed in ECO limit the search space, and the empirical results provide the most accurate information to the compiler to tune performance parameter values.

### 7.3 Configuration Module (CM)

When large datasets are analyzed using complex operations, an analysis workflow may take too long to execute. In such cases, users may be willing to accept lower quality output for reduced execution time, especially when there are constraints on resource availability. The user may, however, desire that a certain application-level quality of service (QoS) be met. Examples of QoS requirements in image analysis include *Maximize the average confidence in classification of image tiles within time*
units and Maximize the number of image tiles, for which the confidence in classification exceeds the user-defined threshold, within $t$ units of time. We have investigated techniques which dynamically order the processing of data elements to speed up application execution while meeting user-defined QoS requirements on the accuracy of analysis. The Configuration Module (CM) draws from and implements the runtime support for these techniques so that accuracy of analysis can be traded for improved performance.

We provide generic support for reordering the data processing operations in our framework by extending Condor’s job scheduling component. When a batch of tasks (such as those produced by expansion of component collections in a Wings workflow instance) is submitted to Condor, it uses a default FIFO ordering of task execution. Condor allows users to set the relative priorities of jobs in the submission queue. However, only a limited range (-20 to +20) of priorities are supported by the `condor_prio` utility, while a typical batch could contain tasks corresponding to thousands of data chunks. Moreover, `condor_prio` does not prevent some tasks from being submitted to the queue in the first place. In our framework, we override Condor’s default job scheduler by invoking a customized scheduling algorithm that executes as a regular job within Condor’s “scheduler universe” and does not require any super-user privileges. The scheduling algorithm implements a priority queue-based job reordering scheme in a manner that is not tied to any particular application. It uses the semantic representations of data chunks in order to map jobs to the spatial coordinates of the chunks. When the custom scheduler decides which chunk to process next based on its spatial coordinates and other spatial metadata properties, it uses this association to determine the job corresponding to this chunk and moves it to the top of the queue.
The custom scheduler can be employed for any application within the spatial data analysis domain. The priority queue insertion scheme can be manipulated for different QoS requirements such that jobs corresponding to the favorable data chunks are scheduled for execution ahead of other jobs. In this way, the customized scheduler helps exercise control over the processing order parameter. When there are no QoS requirements associated with the user query, our framework reverts to the default job scheduler within Condor.

7.4 Experimental Evaluation

In this section, we present an experimental evaluation of our proposed framework using two real-world applications. Our evaluation was carried out across two heterogeneous Linux clusters hosted at different locations at the Ohio State University. The first one (referred to here as RII-MEMORY) consists of 64 dual-processor nodes equipped with 2.4 GHz AMD Opteron processors and 8 GB of memory, interconnected by a Gigabit Ethernet network. The storage system consists of 2x250GB SATA disks installed locally on each compute node, joined into a 437GB RAID0 volume. The second cluster, (referred to here as RII-COMPUTE), is a 32-node cluster consisting of faster dual-processor 3.6 GHz Intel Xeon nodes each with 2 GB of memory and only 10 GB of local disk space. This cluster is equipped with both an InfiniBand interconnect as well as a Gigabit Ethernet network. The RII-MEMORY and RII-COMPUTE clusters are connected by a 10-Gigabit wide-area network connection – each node is connected to the network via a Gigabit card; we observed about 8 Gigabits/sec application level aggregate bandwidth between the two clusters. The head-node of the RII-MEMORY cluster also served as the master node.
of a Condor pool that spanned all nodes across both clusters. A Condor scheduler instance running on the head-node functioned both as an opportunistic scheduler (for “vanilla universe” jobs) and a dedicated scheduler (for parallel jobs). The “scheduler universe” jobs in Condor, including our customized scheduling algorithm, when applicable, run on the master node. All other nodes of the Condor pool were configured as worker nodes that wait for jobs from the master. DataCutter instances executing on the RII-COMPUTE cluster use the MVAPICH flavor of MPI for communication to exploit the InfiniBand interconnect. Our evaluation of the ECO compiler’s automated parameter tuning was carried out independently on a Linux cluster hosted at the University of Southern California. In this cluster (referred to as HPCC) we used a set of 3.2 GHz dual Intel Xeon nodes each with 2 GB of memory for our evaluation.

In the following experiments, we evaluate the performance impact of a set of parameter choices on workflow execution time. First, a set of accuracy-preserving parameters are explored, as we would initially like to tune the performance without modifying the results of the computation. We subsequently investigate the accuracy-trading parameters.

### 7.4.1 Accuracy-preserving Parameters

We used our framework to evaluate the effects of three different accuracy-preserving parameters on the execution time for the PIQ workflow – (i) the chunksize, a component-level parameter, (ii) the task granularity, a workflow-level parameter, and (iii) numActiveChunks, a component-level parameter that is specific to the warp component. Two additional accuracy-preserving parameters – component config and

\[ \text{http://mvapich.cse.ohio-state.edu} \]
component placement were set to their optimal values based on our prior experience with and evaluations of the PIQ workflow. In an earlier chapter [44], our evaluations revealed that certain components of the PIQ workflow such as normalize and autoalign were much faster when mapped for execution onto cluster machines equipped with fast processors and high-speed interconnects. Based on these evaluations and our knowledge of the data exchange between components in the workflow, we determined an optimal component placement strategy which minimized overall computation time as well as the volume of data exchange between nodes. In this strategy, components zproject, prenormalize, stitch, reorganize, warp and the preprocess meta-component execute on the RII-MEMORY cluster nodes, while the normalize, autoalign and mst components are mapped to the faster processors of the RII-COMPUTE cluster. This component placement strategy allows for maximum overlap between computation and data communication between sites for the PIQ workflow. In another earlier work [69], we designed multiple algorithmic variants for the warp and preprocess components of the PIQ workflow. Our evaluations helped us select the most performance-effective variant for these components depending upon the resources they are mapped to. In our experiments, we fixed the values of these latter two parameters and then set out to tune or determine optimal values for the remaining parameters. Our strategy was to treat different parameters independently, selecting a default value for one parameter while exploring the other. The decision as to which parameter to explore first is one that can either be made by an application developer, or can be evaluated systematically by a set of measurements, such as the sensitivity analysis found in [37].
**Effects of chunksize:** For these experiments, we used a 5 GB image with 8 focal planes, with each plane at a resolution $15,360 \times 14,400$ pixels. The chunksize parameter determines the unrolling factor for component collections in the template shown in Figure 7.2(a). Varying the chunksize parameter value affects the structure of the resulting workflow instance and the number of workflow tasks to be scheduled, as shown in table 7.1.

<table>
<thead>
<tr>
<th>chunksize (pixels)</th>
<th># of chunks in a plane</th>
<th># of tasks in workflow (no clustering)</th>
<th># of tasks in workflow (horizontal clustering:::32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$512 \times 480$</td>
<td>900</td>
<td>2708</td>
<td>95</td>
</tr>
<tr>
<td>$1024 \times 960$</td>
<td>225</td>
<td>683</td>
<td>32</td>
</tr>
<tr>
<td>$1536 \times 1440$</td>
<td>100</td>
<td>308</td>
<td>20</td>
</tr>
<tr>
<td>$2560 \times 2400$</td>
<td>36</td>
<td>116</td>
<td>14</td>
</tr>
<tr>
<td>$3072 \times 2880$</td>
<td>25</td>
<td>83</td>
<td>9</td>
</tr>
<tr>
<td>$5120 \times 4800$</td>
<td>9</td>
<td>35</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 7.1: Number of tasks in PIQ workflow instance for a given chunksize parameter value.

The disparity among the number of tasks in the resulting workflow instances will increase as the images grow in size, because larger images can accommodate more combinations of the chunk dimensions. If workflow templates have a larger number of component collections, then the number of tasks in the resulting workflow instance will vary more greatly with chunksize value. As job submission and job scheduling overheads are sizeable contributions to the overall execution time, one possible optimization technique is to employ horizontal clustering of tasks from every component collection. The table also shows the number of tasks in the resulting PIQ workflow instance for each chunksize value when horizontal clustering by Pegasus is used to
group tasks from component collections into bundles of 32 tasks each. Ideally, with horizontal clustering, one should expect diminishing job scheduling overheads and lesser disparity in the execution times observed at different chunksize values.

Figure 7.4(a) shows the overall execution time for the PIQ workflow instance when choosing different chunksize parameter values, both with and without using horizontal job clustering (We used 16 RII-MEMORY nodes and 8 RII-COMPUTE nodes for these experiments). This time is inclusive of the time to stage data in and out of each site during execution, the actual execution times for each component, and the job submission and scheduling overheads. We observe that: (1) At both extremes of the chunksize parameter value range, the execution times are high. The job submission and scheduling overheads for the large number of tasks dominate at smallest chunksize value. At the largest chunksize value, the number of resulting chunks in an image becomes lesser than the number of worker nodes available for execution. Since each task must process at least one chunk, the under-utilization of resources leads to higher execution times. (2) The intermediate values for chunksize yield more or less similar performance, except for an unexpected spike at chunksize = 2560 × 2400. On further investigation of the individual component execution times, we observed that the algorithmic variant used for the warp component – which accounts for nearly 50% of the overall execution time of PIQ – performed poorly at this chunksize value. Figures 7.4(b) and 7.4(c) show how chunksize value affects performance at the level of each component. (3) Horizontal job clustering, as expected, lowers execution time at smaller chunksize values. At larger chunksize values, the lack of a large job pool to begin with renders job clustering ineffective.
Figure 7.4: Effect of varying chunksize parameter on PIQ workflow execution
As mentioned earlier, chunksize value is expected to have a greater impact on the performance for larger images because of the larger range within which the parameter values can be varied. Figure 7.4(d) presents our evaluation results obtained for a 17 GB image (with 3 focal planes, each plane having $36864 \times 48000$ pixels). We used 32 RII-MEMORY nodes and 8 RII-COMPUTE nodes for these experiments. The chunksize parameter has been shown using separate axes for the chunk width and the chunk height for better viewing of the results. The surface was plotted based on results obtained using 75 different values of the chunksize parameter. Again, we observe poor performance at the extreme low and high values of chunksize for the same reasons outlined earlier. Here, we also observed that chunksize values that correspond to long horizontal stripe chunks yielded better performance for the PIQ workflow and this class of data. This tells us that most analysis operations in the PIQ workflow favor horizontal striped chunks. In future endeavors, our framework will seek to determine the best values of the chunksize parameter by efficient navigation of the surface based on sampling techniques and trends generated from training data.

**Effects of Task Granularity**: In these experiments, we coalesced components of the PIQ workflow into meta-components to produce a workflow template with a coarser task granularity. Figure 7.5 illustrates an alternative workflow template for the PIQ application generated using Wings, that groups components into meta-components. Here, the *zproject* and *prenormalize* steps from the original template are fused to form *metacomp1*; *normalize, autoalign, mst* are collectively *metacomp2* while *stitch, reorganize, warp* form *metacomp3*. *Preprocess* is the fourth meta-component in the
Figure 7.5: Wings workflow template for the PIQ application with meta-components
workflow template and includes the thresholding, tessellation and prefix sum generation components. By using this representation, our framework further reduces the number of tasks in a workflow instance. When component collections are embedded within a meta-component, they are not explicitly unrolled at the time of workflow instance generation. Instead, they are implicitly unrolled at runtime within a corresponding DataCutter instance. That is, the chunksize input parameter to a meta-component is propagated to the DataCutter filters set up for each component within that meta-component. Based on the value of chunksize, DataCutter will create multiple transparent copies of filters that handle the processing of tasks within a component collection. Each such filter copy or instance will operate on a single chunk at a time. We also manipulated the execution strategy within the preprocess meta-component to support task-parallelism (i.e., pipelined execution of data chunks) in order to avoid disk I/O overheads for the meta-component.

![Execution time with different task granularity](image)

Figure 7.6: Execution time with different task granularity (5GB image, 40 nodes, chunksize=512 x 480)
Figure 7.6 shows that the overall execution time for the PIQ workflow improves by over 50% when the task granularity strategy employed includes meta-components. The figure also shows the changes in execution times for individual components of the workflow. For parallel components like *autoalign* and *warp*, there is no difference in performance because the actual scheduling of processes within a component is carried out in the same way regardless of whether meta-components are used or not. However, component collections like *zproject*, *normalize* and *reorganize* benefit from the high-granularity execution. Depending on the nature of the tasks within a collection, our framework can use explicit unrolling and execution via Pegasus, or implicit unrolling within a meta-component and execution via DataCutter.

**Effects of parameter tuning for individual components:** In these experiments, we take a closer look at accuracy-preserving parameters that are specific to individual components in the workflow. Our goal here is to show how the autotuning of such parameters based on model-guided optimization techniques within the ECO compiler can reduce the execution times of the individual components, and hence, of the overall workflow. In the PIQ workflow, one of the algorithmic variants of the *warp* component, known as the On-Demand Mapper (ODM), was shown to provide best performance among all variants, as long as the amount of physical memory available on the cluster nodes was not a limiting factor in the execution [69]. The benefits of ODM derived from the fact that ODM, unlike the other variants, seeks to maintain large amounts of data in memory during execution. We identified a parameter called *numActiveChunks* that affects the execution time of the ODM variant by limiting the number of ‘active’ data chunks that ODM can maintain in memory on a node at any
given time. The larger the number of active chunks that can be maintained in memory, the greater the potential for data reuse when the warping transformations are computed for the input data. The potential reduction in chunk reads and writes via data reuse leads to lesser execution time. The maximum value of \texttt{numActiveChunks} parameter depends on the size of the chunks and the available physical memory on a node. However, the optimal \texttt{numActiveChunks} value may depend on other factors such as the potential for data reuse in the ODM variant for a given memory hierarchy which need to be modeled appropriately.

We introduced a novel technique for modeling that discovers the behavior of such application-level parameters through the use of functional models and statistical methods. Our technique requires the component developer to identify a set of integer-valued parameters, and also specify the expected range for the optimal parameter value. The models are derived from sampling the execution time for a small number of parameter values, and evaluating how well these sample points match a functional model. To derive the functional model of the sample data, we use the curve fitting toolbox in MATLAB to find the function that best fits the data, and also to compute the $R^2$ value in order to quantify the accuracy of our model. Ultimately, we discovered that the double-exponential function ($y = a e^{bx} + c e^{dx}$) was the best overall match for the application-level parameters in our set of experiments. Using this double-exponential function model, the algorithm can select the parameter value for which the function is minimized, and can dynamically identify the neighborhood of the best result.

Our experiments were conducted on nodes of the HPCC cluster. Here, we evaluated only the \textit{warp} component of the PIQ workflow. Specifically, we evaluated only
the ODM variant of the component in order to determine the optimal value of the \texttt{numActiveChunks} parameter. Figure 7.7 shows results of our evaluation from warping a single slice of the image data described earlier in this section. We try to model the behavior of the \texttt{numActiveChunks} parameter for the input data using a statistical analysis approach. In this figure, the curve represents the double-exponential function obtained using five sample data points, while the remaining points obtained from the exhaustive search within the range of permissible parameter values are also shown. The area between the dotted lines represents the range of parameter values that our model predicts will have performance within 2% of the best performance. In this case, we predicted the optimal value of 74 for the \texttt{numActiveChunks} through the function model, which ended up being within 1.25% of the real best performance that one could obtain using the exhaustive search.

Figure 7.7: Autotuning to obtain optimal value of \texttt{numActiveChunks} parameter for the \texttt{warp} component; 8 processors
We validate our model by comparing against performance results from an exhaustive set of experiments across the entire permissible parameter range. Overall, we have achieved speedups up to 1.38X as compared to the worst-case performance from an exhaustive search while being within 0.57% and 1.72% of the best performance from the exhaustive search. We examined only 5% of the parameter search space by computing five point samples. Our experimental results show speedups of up to 1.30X as compared to the user-specified default parameter value. While our speedup is small when placed in the context of the overall PIQ workflow, our selection of parameter values based on a statistical analysis approach derives improved performance as compared to current user-specified values.

7.4.2 Accuracy-trading Parameters

These experiments demonstrate performance gains obtained when one or more accuracy-trading parameters were modified in response to queries with QoS requirements. The optimizations in these experiments are derived from the custom scheduling approach described in section 7.3 that makes data chunk reordering decisions dynamically as and when jobs are completed. In all our experiments, we observed that the overall execution time using our custom scheduling algorithm within Condor is only marginally higher than that obtained from using the default scheduler, thereby showing that our approach introduces only negligible overheads. These experiments were carried out on the RII-MEMORY cluster 43 with Condor's master node running our customized scheduler. We carried out evaluations using multiple images

43Our goal here is only to demonstrate our framework’s ability to exploit performance-quality trade-offs, and not adaptability to heterogeneous sets of resources. We note that this experiment could also be carried out on the same testbed used for the PIQ application.
(ranging in size from 12 GB to 21 GB) that are characterized by differences in their data (feature) content. We target user queries with two kinds of QoS requirements:

**Requirement 1:** Maximize average confidence across all chunks in an image within time $t$; **Requirement 2:** Given a confidence in classification threshold, maximize the number of finalized chunks for an image within time $t$. These requirements can be met by tuning combinations of one or more relevant accuracy-trading parameters.

**Tuning only the processing order parameter for requirement 1:** This is useful when users wish to maximize average confidence across an image while processing all chunks at the highest resolution, i.e. trading quality of result for the overall image, but not the quality of the result for individual chunks. In such cases, the processing order parameter can be tuned such that the Condor customized scheduler prioritizes chunks that are likely to yield output with higher $(\text{confidence/time})$ value at the maximum resolution.

![Graphs](image)

(a) Quality Improvement by tuning only the processing order parameter

(b) Quality Improvement by tuning the processing order and resolution parameters

**Figure 7.8:** Using accuracy parameter tuning to improve response under time constraints
Figure 7.8(a) shows that a tuned parameter helps achieve a higher average confidence at maximum resolution across all chunks. For example, after 800 seconds, the tuned parameter execution achieves an average confidence of 0.34 which is greater than the 0.28 value achieved when no parameter tuning is employed. This is because our customized scheduler tunes the processing order parameter to reorder chunk execution in a manner that favors jobs corresponding to chunks that yield higher confidence.

**Tuning both the resolution and processing order parameters for requirement 1:** This is useful when individual chunks can be processed at lower resolutions so long as the resulting confidence exceeds a user-specified threshold. Figure 7.8(b) shows how parameter tuning helps achieve higher average confidence at all points during the execution. Each chunk is iteratively processed until only that target resolution at which the confidence exceeds a threshold (set to 0.25 here). Our custom scheduler prioritizes chunks that are likely to yield output with higher \( \frac{\text{confidence}}{\text{time}} \) value at lower resolutions.

**Tuning both the resolution and processing order parameters for requirement 2:** Here, the customized scheduler prioritizes jobs corresponding to chunks that are likely to get finalized at lower resolutions. Figure 7.9(a) shows how parameter tuning in our framework yields an increased number of finalized chunks at every point in time during the execution. The improvement for this particular case appears very slight because the confidence in classification threshold was set relatively high as compared to the average confidence values generated by the classify component, and this gave our custom scheduler lesser optimization opportunities via reordering.
Figure 7.9: Quality and scalability tests for requirement 2

Scalability: In this set of experiments (carried out as part of requirement 2), we scaled the number of worker nodes in our Condor pool from 16 to 48. Our goal here was to determine if our custom scheduler could function efficiently when more worker nodes are added to the system. Figure 7.9(b) shows how the time taken to process a certain number of chunks in an image halves as we double the number of workers. Hence, the scheduler performance scales linearly when an increasing number of resources need to be managed.

7.5 Summary

Our current implementation of the proposed framework provides support for users to manually express the values of the various performance parameters in order to improve performance. Our observations include:
• The experimental evaluation of the proposed framework shows that adjustments of accuracy-preserving and accuracy-trading parameters lead to performance gains in two real applications.

• The framework also achieves improved responses to queries involving application-level quality of service (QoS) requirements.

• To achieve the performance gains, users only need to provide high-level specifications for the application. In other words, different sets of users together with domain knowledge is used to collectively represent the execution context using rich semantic metadata descriptions. The integrated middleware suite is then configured suitably to carry out the execution plans.

• The customized Condor scheduling module facilitates a mechanism for trading analysis accuracy for performance with user-defined quality of service requirements.

We target application scenarios where a given workflow is employed to process a large number of data elements (or a large dataset that can be partitioned into a set of data elements). In such cases, a subset of those data elements could be used to search for suitable parameter values (by applying sampling techniques to the parameter space) during workflow execution and subsequently refining the choice of parameter values based on feedback obtained from previous runs. Statistical modelling techniques similar to those used in the Network Weather Service [112] can be used to predict performance and quality of future runs based on information gathered in previous runs.
CHAPTER 8

CONCLUDING REMARKS

This chapter summarizes the contributions of this thesis and suggests possible directions for future research that can be built on these contributions.

In this thesis document, I have motivated the need for a paradigm of "specification, configuration and execution" to address the performance requirements of scientific applications that analyze very large volumes of high-resolution, multi-dimensional datasets. Such applications are typically structured as workflows of several components or tasks that either do low-level data transformations and processing or high-level analysis and data modeling. Macro-scale studies within different domains involve analysis carried out across entire datasets (e.g., whole-slide imaging) at once. These data-intensive applications need to perform complex analysis operations across large amounts of disk-resident data in a distributed shared-nothing architecture system. Carrying out these analysis tasks within reasonable turnaround time for results is a challenging problem. Current middleware layers rely extensively on scale-out parallelism and are not flexible enough to meet the challenges. Hence, execution strategies for data-intensive scientific applications needed further investigation to address performance requirements.
My research draws inspiration from frameworks like High Performance Fortran (HPF) where the existing application description in the Fortran array language is embellished with user-provided high-level directives aimed at improving efficiency – the user specifies only high-level constructs for data distribution and the compiler takes care of the tedious message-passing code specific to the architecture. A large part of my research was spent on identifying a set of directives in the context of data-intensive scientific applications that analyze high-resolution multi-dimensional datasets. In my research, I explored the following different avenues to improve end-to-end performance of application workflows with varying degrees of success.

- First, I focused on minimizing the disk I/O overheads for low-level transformation tasks as they are primarily I/O-intensive and the disk bandwidth even on modern computing systems is relatively slow when compared with other commodity components. I investigated two approaches to designing and implementing a set of multi-dimensional data services to assist applications in storage and retrieval of data chunks in a distributed environment. The first approach targeted peer-to-peer storage systems that are re-emerging in the context of large federated Cloud-based storage and NoSQL database systems. This was an implicit approach because the application sought to provide hints to the system to configure its data replication policy by scheduling data requests in specific ways. However, the poor scalability of the peer-to-peer prototype system prevented us from making observations at desired data scales.

The second approach targeted storage systems compose of local disks on active storage nodes of high-end clusters. A set of mediator services provided a layer of I/O virtualization to application programs. This was an explicit approach where
applications could configure the mediator services to manage the way data is stored on disks. The mediator approach allowed us to scale linearly to handle multi-terabyte sized datasets. Through careful configuration on a mass-storage cluster, we were able to attain up to 91% of the peak application-perceived collective disk I/O bandwidth on the cluster.

- Second, going beyond scalability, I developed a layer of application-level configuration services to resolve mismatches between analysis requirements and system properties brought about by the workflow scheduling algorithm used. Experimental evaluations for three different application use-case scenarios at large data scales showed that identifying and resolving mismatches through application-level configuration led to substantial improvements in end-to-end performance, and helped meet time constraints.

- Third, I focused on minimizing the execution time of adaptive, high-level analysis tasks that perform statistical computations to produce output and the accuracy of analysis. When scalability and other configurations that preserve the quality of the result do not provide the desired performance, we target the adaptive nature of such tasks and the opportunities they provide. I explored the trade-offs between accuracy of analysis and performance when processing large multi-dimensional datasets. As predicted via extensive simulation studies, our object-based heuristic was shown to provide improved response to user-specified application-level QoS requirements. However, real life applications exhibited characteristics due to which our initial assumptions did not hold. A second region-based heuristic that uses spatial locality of feature content in the data
provided up to a 40% improvement over a strategy that does not configure any parameters.

My work up until this point was based on careful understanding and configuration of legacy applications for performance gains on a variety of systems. The remainder of the thesis focused on discovering ways to represent the knowledge gained from my experiences in a concise manner to provide high-level configuration-as-a-service to users of new applications.

- My next contribution is to provide a high-level specification layer that supports the expression of conceptual knowledge about the application domain. An important part of the conceptual knowledge is the description of the configurability requirements for applications in terms of concepts and properties defined for the domain. For any new application instance within our target domain, the availability of dedicated configuration and execution services on our system means that end-users only need to provide information about the application at a high level of abstraction. Domain-specific configuration-as-a-service will then map the requirements of the new application onto the appropriate services for a given system.

- A prototype implementation of our proposed framework was demonstrated for the manual configuration of a set of accuracy-preserving and accuracy-trading parameters in two real applications. Our experimental results from evaluations conducted at large data scales serve as a proof of concept for our proposed
approach of high-level specification for application-driven, system-directed con-
figuration with support for accuracy trade-offs and other runtime performance
optimizations during execution.

While my thesis has focused on the domain of large-scale multi-dimensional data
analysis applications, there are other classes of applications borrowed from the en-
terprise computing world that share similar motivating requirements. The use of
best-effort computing to meet the demands of such applications by appropriately
configuring parallel hardware and software is gaining traction [29]. The authors focus
on providing language and runtime support to carry out these accuracy trade-offs.
The use of semantically rich metadata descriptions of the application execution con-
text, where different users express conceptual knowledge through ontologies and rules
holds promise for other application domains as well.

While the proposed IMPACT framework provides mechanisms for high-level de-
scriptions and directives, the process of configuring the parameters for execution is
still predominantly a manual process. Automated runtime optimizations can be ap-
plied in few cases where the additional knowledge required is either provided by the
user or can be predicted during execution. Further automation is required to have
the framework configure all parameters without human intervention. The IMPACT
framework could be augmented with a performance-data gathering and mining frame-
work that statistically analyzes prior history of execution to configure parameters for
future runs.

As users can introduce newer performance and accuracy parameters through the
specification module, and also support their scalable execution, this opens up the
avenue for research into many compiler optimization techniques to determine the appropriate configuration for those parameters on a given system architectures.
BIBLIOGRAPHY


[92] Ioan Raicu, Ian Foster, Mike Wilde, Zhao Zhang, Kamil Iskra, Peter Beckman, Yong Zhao, Alex Szalay, Alok Choudhary, Philip Little, Christopher Moretti, Amitabh Chaudhary, and Douglas Thain. Middleware support for many-task computing. Cluster Computing, 13(3):291–314, 2010.


