Using the Architectural Tradeoff Analysis Method to Evaluate the Software Architecture of a Semantic Search Engine: A Case Study

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Sandeep Chatra Raveesh

Graduate Program in Computer Science and Engineering

The Ohio State University

2013

Master's Examination Committee:

Dr. Jayashree Ramanathan, Advisor

Dr. Rajiv Ramanath
Abstract

The software architecture greatly determines the quality of the system. Evaluating the architecture during the early stage of development can reduce the risk. When used appropriately, it will have a favorable effect on the system. Architectural Tradeoff Analysis Method (ATAM) is an architecture evaluation technique for understanding the tradeoffs in the architecture of software systems. This thesis describes the application of ATAM to evaluate the query engine component of the ResearchIQ.

ResearchIQ is a semantically anchored resource discovery tool, which will help the researchers in the domain of clinical and translation science to discover the resources in a simplified manner. The primary goal of ResearchIQ is the delivery of search results effectively to the researchers.

A large part of thesis is devoted in evaluating the architectural alternatives of the query engine component of ResearchIQ using ATAM. Three initial architectures alternatives are presented in the thesis. The thesis introduces the system (ResearchIQ) being evaluated, its business drivers and background. It also provides a general overview of the ATAM process describes the application of the ATAM to the ResearchIQ system and presents the important results. The document is intended as a report to develop a
prototype implementation that led to the final framework that enhances the performance of ResearchIQ.
Dedication

This document is dedicated to my family and friends.
Acknowledgments

A special thanks you to my advisors, Dr. Jayashree Ramanathan and Dr. Rajiv Ramnath for their guidance throughout this project and my graduate studies. I have benefited greatly from your continual support and direction.

I would also like to acknowledge Dr. Philip Payne, Dr. Tara Payne, Omkar Lele and the rest of the ResearchIQ team; Puneet Mathur and Satyajeet Raje, without their contributions this work would not have been possible. Likewise, I thank CETI and my colleagues at the CSE Department.

This work was done under the aegis of the Center for Experimental Research in Computer Science at Georgia Tech and The Ohio State University, funded by the National Science Foundation's I/UCRC program (NSF IIP 0753710). Funding was also provided by the National Science Foundation Course, Curriculum, and Laboratory Improvement program (NSF CCLI Phase 0837555). I wish to acknowledge the helpful ideas by CETI students, colleagues, members, and sponsors.
# Table of Contents

Abstract ............................................................................................................. ii

Dedication ........................................................................................................ iv

Acknowledgments ............................................................................................ v

Table of Contents ........................................................................................... vi

List of Tables .................................................................................................... viii

List of Figures .................................................................................................. ix

Chapter 1: Introduction ..................................................................................... 1

Chapter 2: Introduction to ResearchIQ ............................................................. 6

Chapter 3: Architectural Tradeoff Analysis Method ....................................... 11

Chapter 4: Scenario Generation and Prioritization ....................................... 14

Chapter 5: Proposed Architectures ................................................................. 18

Chapter 6: Scenario Analyses ......................................................................... 28

Chapter 7: Critique of the Options ................................................................. 35

Chapter 8: Sensitivity Analyses ..................................................................... 36

Chapter 9: Conclusion ..................................................................................... 37
List of Tables

Table 1: Quality attribute utility tree ................................................................. 15
Table 2 Query latency comparison ................................................................. 28
Table 3 Cache Memory comparison ............................................................... 30
Table 4 Preprocessing time ............................................................................. 32
List of Figures

Figure 1: ResearchIQ Component Diagram ..............................................................8
Figure 2 Home Page with Auto complete list ............................................................10
Figure 3 Steps of ATAM .........................................................................................12
Figure 4 Utility Tree ...............................................................................................14
Figure 5 Basic Architecture ...................................................................................21
Figure 6 Full Data Cache .......................................................................................24
Figure 7 Reduced Data Cache ...............................................................................27
Figure 8 Time taken vs. Total query result ............................................................29
Figure 9 Memory size vs. Query terms ...................................................................31
Figure 10 Hadoop based preprocessing ..................................................................33
Figure 11 Preprocessing time vs. Total query terms ...............................................34
Chapter 1: Introduction

Background

Architecture tradeoff analysis method (ATAM) [1] is a technique for evaluating the software architecture. It is used early in the software development cycle to help choose a suitable architecture. ATAM helps in understanding the tradeoffs in the architecture of software systems. In this thesis, ATAM is used to evaluate the query engine component of ResearchIQ [2].

ResearchIQ is a semantically anchored resource discovery tool [3], aimed at helping the researchers in the domain of clinical and translation science to discover the resources in a simplified manner. It facilitates the discovery of semantically related local and publicly available data through a single web portal. ResearchIQ makes use of the ontological framework for the semantic annotation of data. The user interface of ResearchIQ is simple and efficient and it is designed using Google Web Toolkit (GWT) [4].
Challenges

Semantic technologies tend to be slower than their syntactic counter parts due to the overhead of additional reasoning required. The primary goal of ResearchIQ is the delivery of semantic search results effectively to researchers. Hence, the speed of the result delivery is a critical factor. A semantic search engine is of not much use if it takes too much time to retrieve and display the results. It is a challenge to implement an efficient search engine that delivers the results quickly. This thesis mainly focuses on the architecture of query engine component of ResearchIQ which is the critical component for the efficiency of search engine.

Contributions of the Study and Thesis

This contribution of this thesis is the prototype implementations that led to the final framework that enhances the performance of a semantic search engine. The implemented framework is made up of a caching layer which is implemented by preprocessing the result set in a distributed environment. In this work, Architectural Tradeoff Analysis Methodology (ATAM) has been used to analyze and evaluate alternative software architectures for the query component of ResearchIQ. ATAM is used early in the software development cycle to help choose a suitable architecture. ATAM helps in understanding the tradeoffs in the architecture of software systems. The framework provides a software architecture that is generic to all the semantic services.
Chapter 2: Related work

This section surveys the previous work related to performance optimization of a search engine by building inverted index and by parallel processing of data. This forms the basis for the proposed architecture presented in this document. This section also includes a brief survey of previous work done in Architectural Tradeoff Analysis Method. ATAM is used to evaluate the software architectures for the query component of ResearchIQ.

Performance optimization of search engine

Google's web search application comprises of several tens of terabytes of uncompressed data [5, 6]. Google has built an inverted index, which maps word to the document identifiers. The inverted index resulting from this raw data is itself terabytes of data. In the first phase, document identifiers are retrieved and in the second phase, for every document identifier, its actual title and URL are computed. Google answers most queries in between 1 to 10 seconds. The infrastructure consists of large clusters which are made up of desktop class machines rather than small number of large scale machines. A similar idea could be used for a semantic search engine [7, 8] like ResearchIQ to make it fast and efficient.
Hadoop is an open source software framework that supports processing of large data sets in a distributed environment. It is a part of Apache project, licensed under the Apache v2 license. It supports, running of applications on large clusters of commodity hardware. It is designed to scale up to thousands of nodes. Hadoop has two main components: MapReduce [7] and Hadoop Distributed File System (HDFS) [8]. MapReduce is a programming model for processing large data in a cluster of commodity hardware. HDFS is a distributed, scalable and reliable Java based file system for the Hadoop framework. Many real world tasks are expressible in this model.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.
Importance of ATAM

The SEI Architecture tradeoff analysis method (ATAM) is a proven technique for systematically evaluating the software architecture for fitness of purpose. It is used early in the software development cycle to help choose a suitable architecture. ATAM helps in understanding the tradeoffs in the architecture of software systems. It exposes the risks that would restrain the achievement of the product. ATAM presents the quality attribute requirement of the architecture and a precise statement of all the proposed architectural designs and refines them. It also evaluates the architectural designs to determine if they meet the quality attribute requirements.
Chapter 3: Introduction to ResearchIQ

ResearchIQ is a semantically anchored resource discovery tool, aimed at helping the researchers in the domain of clinical and translation science to discover the resources in a simplified manner. It facilitates the discovery of semantically related local and publicly available data through a single web portal. ResearchIQ makes use of the ontological framework for the semantic annotation [10] of data. The user interface of ResearchIQ is simple and efficient and it is designed using Google Web Toolkit (GWT).

ResearchIQ provides semantic search capabilities over several heterogeneous data sources. A list of the type of resources is given below:

**PubMed**

PubMed comprises more than 22 million citations for biomedical literature. Currently ResearchIQ holds over 14000 articles from PubMed. These articles are related to The Ohio State University from the year 2004 till present (April 2013).

**OSUPro**

OSUPro contains the profiles for faculty and researchers within The Ohio State University. ResearchIQ holds over 1200 OSUPro profiles related to biomedical informatics.
Clinical Trials
ResearchIQ currently holds over 950 publically available clinical trials dataset under clinicaltrials.gov and the studies registered under the StudySearch database.

Government Grants
ResearchIQ holds over 1600 government grant proposal information from various funding sources such as NIH, Department of public health, etc.

Websites
The Ohio State Medical Center has various labs, departments, resources and equipment.
Research holds around 300 websites, which contains this information.

Components of ResearchIQ

Figure 1. shows the different components of the system. There are three main components: Meta-Engine, Query Engine and the User Interface.

Meta-Engine
The Meta-Engine is a data extraction and semantic annotation tool. The Extractor parses the data from various sources and provides the annotator with the raw data. It also captures the metadata related to each of the resource like URL, preferred label, etc. The Annotator generates the semantic view of the raw data in the form of Resource
Description Framework (RDF) triples, a W3C recommended framework for data exchange. It uses the MetaMap, a free text annotator for the biology and medical domains to generate the semantic view of the resources.

Figure 1: ResearchIQ Component Diagram
Query Engine

The semantic metadata generated by the Meta-Engine is stored in a semantic triple store in the RDF format. The query engine runs the semantic queries on triple store. Query engine makes use of SPARQL query language to query the triple store. SPARQL, one of the key technologies of the semantic web, is an RDF query language that is able to retrieve and manipulate the data stored in RDF format.

User Interface

The user interface for ResearchIQ is designed using GWT (Google Web Toolkit). GWT is an open source development toolkit that allows web developers for building and optimizing complex browser-based applications. It is designed in such a way that it can effectively deliver the search results to the end user. For the purpose of completeness, screenshots of the user interface is provided in the Figure 2.
Figure 2 (a) Home Page with Auto complete list. (b) Search Results for “Bio-informatics”
Chapter 4: Architectural Tradeoff Analysis Method

Architecture tradeoff analysis method (ATAM) is a technique for evaluating the software architecture. It is used early in the software development cycle to help choose a suitable architecture. ATAM helps in understanding the tradeoffs in the architecture of software systems. The major goals of the ATAM are to

- Present the quality attribute requirement of the architecture.
- Present a precise statement of all the proposed architectural designs and refine them.
- Evaluate the architectural designs to determine if they meet the quality attribute requirements.

The Steps in ATAM

ATAM is divided into four main phases: scenarios and requirement gathering, architectural views presentation, scenario realization, and sensitivity analysis and tradeoffs. Each of these phases is explained briefly below.

Phase 1 – Scenario and requirements gathering

The first phase in ATAM involves the project manager presenting the business drivers to all the participants. The participants will build a utility tree. And then, participants decide the most important quality attributes that need to be considered for this architecture.
Based on these quality attributes, participants will elicit a large set of scenarios that expresses those quality attributes. This set of scenarios is then prioritized involving the entire stakeholder group. Apart from the scenarios, constraints and environments of the system must be identified.

Figure 3 Steps of ATAM
Phase 2 – Architectural views presentation

The second phase involves the architect identifying the architectural approaches considering the requirements, scenarios and software design principles. He then elicits the architectural approaches to all the participants. Using the ATAM is a continual process of choosing the software architecture, which will meet to required quality attributes generated by all the participants.

Phase 3 – Scenario realization

After the requirements, constraints and environment are gathered, architect will map the quality attributes to the different architectures elicited to see how it responds to each scenario. Each quality attribute must be analyzed in isolation, with respect to all architectures elicited in the previous phase.

Phase 4 – Sensitivity analysis and tradeoffs

The fourth phase involves identifying the sensitivity points. If the architecture affects any quality attribute then it is considered to be the sensitivity point. If it affects more than one quality attributes then it is considered as tradeoff point.

Based on all the information collected in the four phases, the ATAM team presents the findings to the assembled stakeholders.
Chapter 5: Scenario Generation and Prioritization

The phase 1 of the ATAM begins with construction of utility tree to elicit and prioritize the quality attributes. Figure 4 represents the utility tree for this system. The top level of tree is the overall utility of the system. The second level is the list of quality attributes with sub divisions that are important to this system. Performance, Availability, Adaptability and Affordability are the quality attributes for this system. Each quality attribute can have further sub divisions. For example, Affordability is further divided into “Total memory required” and “Total preprocessing time”. The leaves (not shown in the Figure 4) represent the scenarios for the quality attributes.

![Utility Tree Diagram]

Figure 4 Utility Tree
# Phase 1: Quality Attribute Utility Tree

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Concerns</td>
<td>A. Minimize query latency</td>
</tr>
<tr>
<td>Scenarios</td>
<td>1. Query latency must be lesser than 5 seconds for result set &lt; 1000</td>
</tr>
<tr>
<td></td>
<td>2. Query latency must be lesser than 25 seconds for result set &lt; 10000</td>
</tr>
<tr>
<td></td>
<td>3. Query latency must be lesser than 1 second for result set = 0</td>
</tr>
<tr>
<td>Stakeholder Vote</td>
<td>User, Business</td>
</tr>
<tr>
<td>Attribute Concerns</td>
<td>B. Minimize preprocessing time</td>
</tr>
<tr>
<td>Scenarios</td>
<td>1. Preprocessing time must be lesser than 10 hours.</td>
</tr>
<tr>
<td>Stakeholder Vote</td>
<td>Business, Developer</td>
</tr>
<tr>
<td>Quality Attribute</td>
<td>Availability</td>
</tr>
<tr>
<td>Attribute Concerns</td>
<td>A. Minimize system down time</td>
</tr>
<tr>
<td>Scenarios</td>
<td>1. System should not be unavailable for more than 60 min per year.</td>
</tr>
</tbody>
</table>

Table 1: Quality attribute utility tree
<table>
<thead>
<tr>
<th>Stakeholder Vote</th>
<th>User</th>
<th>Quality Attribute</th>
<th>Af- fordability</th>
<th>A. Ability to manage the required hardware cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Attribute Concerns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenarios</td>
<td>1. Hardware cost should be minimal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stakeholder Vote</td>
<td>Business</td>
<td>Quality Attribute</td>
<td>Modularity</td>
<td>A. Replace an architectural component</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Attribute Concerns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenarios</td>
<td>1. Replace the NoSQL database during the maintenance and accomplish replacement within 1 person week.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stakeholder Vote</td>
<td>Developer</td>
<td>Quality Attribute</td>
<td>Adaptability</td>
<td>A. Ability to accommodate new requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Attribute Concerns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenarios</td>
<td>1. Add new metadata to the resources.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stakeholder Vote</td>
<td>Developer</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For the ResearchIQ evaluation, the evaluation team chose three of the highest priority scenarios and analyzed them.

The three highest priority scenarios chosen are listed below:

- Minimize the query latency
- Minimize the memory cost
- Minimize the preprocessing time
Chapter 6: Proposed Architectures

Definitions

Resource

A resource in ResearchIQ is a source of data, knowledge or tools. The types of resources are publications, web pages, expertise, clinical data sets, medical data sets, etc. The details of the variety of these resources are provided in the contributions section. The goal of ResearchIQ is to retrieve these resources using semantic search logic. Every resource consists of the following metadata.

- **URI**: Uniform resource identifier (URI) of the resource is unique across the resources. Resources are identified by their URI.
- **URL**: The Uniform resource locator (URI) is the source location of the resource.
- **Description**: A short description of the resource that will be displayed to the user on the user interface.
- **Resource type**: Identifies the type of the resource.
- **Score**: Is the measure of relative importance for the query term. It is dynamic.
- **Path**: Path gives the all the nodes, which lead to this resource.
 Technologies and Tools

**Lucene/Solr**

Apache Lucene [15] is an open source information retrieval software library. It is an indexing and search library that also provides additional capabilities like spellchecking, hit highlighting, etc. SOLR [16] is built using Lucene core. It is a high performance search server. SOLR supports hit highlighting, faceted search, caching, and replication and it also comes with a web admin interface.

**Couchbase server**

Couchbase server [17] is an open source NoSQL, document-oriented database. Couchbase provides a simple node management system. It simplifies the management of cluster by providing a web interface. Nodes can be added or removed in one click from web interface. Couchbase also supports the auto-sharding, which will balance the load automatically across cluster. Couchbase is built using Memcached, an in-memory key value store, which is used as a caching layer. It can be used as a replacement for memcached layer. It helps in dealing with the memcached problems like cold cache, lack of scale out flexibility, etc.

**Hadoop**

Hadoop [18] is an open source software framework that supports processing of large data sets in a distributed environment. It is a part of Apache project, licensed under the Apache v2 license. It supports the running of applications on large clusters of commodity
hardware. It is designed to scale up to thousands of nodes. Hadoop has two main components: MapReduce and Hadoop Distributed File System (HDFS). MapReduce is a programming model for processing large data in a cluster of commodity hardware. HDFS is a distributed, scalable and reliable Java based file system for the Hadoop framework.

**Triple Store**

A triple store is a database for the storage and retrieval of triples. A triple is a data entity composed of subject-predicate-object. For example “New Delhi is the capital of India”. Here, “New Delhi” is the subject, “capital of” is the predicate and “India is the object”. A triple store is optimized for the storage and retrieval of triples. Triples within the triple store can be accessed using query language, much like a relational database.

**Architecture Alternatives**

**Basic Architecture**

Query pipeline of ResearchIQ is as shown in Figure 5. Query can consist of multiple individual terms. First step is to separate these terms. These individual terms will be sent to the query engine to fetch the results. Query engine has two main components: Seed Resource Finder and Score Propagator.
Figure 5 Basic Architecture
Seed Resource Finder

Seed finder uses the Lucene API’s to search for the seed resources within the Lucene indices. The output of this stage will be a set of seed resources and its associated scores. These resources will be added to the result set.

Score Propagator

Score Propagation is a semantic resource finding mechanism. Seed resource scores generated by the Seed Resource Finder will be used by the Score Propagator to propagate the score and find the additional resources in the Triple Store. Resultant resources are added to the final result set. The detailed process of querying is given below.

Initially the query term will be split into individual terms. Then for each term, results will found separately and then they will be merged before displaying results in web portal.

Seed score generation:

- For every query term, we generate all its synonyms.
- A set of seed resources and its associated scores are generated using each synonym.
- Seed resources are propagated using the propagation technique, which is explained, in the next section.
- The result from the previous step is added a set called “resultset”.
- Scores are then normalized with respect to the highest score from each resultset.
Challenges

The main disadvantage with basic architecture mentioned in the previous section is the query latency. The primary goal of ResearchIQ is the delivery of search results effectively to researchers. Hence, the speed of the result delivery is a critical factor. A simple solution to this problem is by introducing a caching layer. Query engine will cache the results before displaying in web portal. Every time the query engine will check to see if the term exists in the cache before querying the triple store. This forms the basis for the next two architectures.

Full Data Cache Architecture

Full Data Cache Architecture has two main components: preprocessing the queries and querying.

Preprocessing the queries

In the preprocessing phase, all the possible query terms are processed beforehand using the basic architecture presented in Figure 5. The final results are stored as it is in a cache. Couchbase server is used as a cache, which is a key value store. Query term is the key and list of resources (result set) is the value.
Querying

Figure 6 represents the querying part in Full Data Cache architecture. It has a very simple query engine which searches for the term in the preprocessed cache. The result will be returned to the user.

Figure 6 Full Data Cache
Reduced Data Cache Architecture

Full Data Cache Architecture described in the previous section consumes large amount of cache memory. Reduced Data Cache Architecture overcomes the problem by reducing the cache memory. It has two main components: preprocessing and querying:

Preprocessing the queries

In the preprocessing phase, all the possible query terms are processed beforehand like the Full Data Cache Architecture; however the way they are stored in cache will be different here. Every resource within the final result set will be split into two parts: Static data and Dynamic data.

Static data

Static data is the metadata associated with every resource. URL, description, preferred label, alternate label, resource type are the static data. The values of these will remain same for a resource irrespective of the query. Thus it is not necessary to store these values for every query. They can be stored once in the cache and then the same values could be used for every query.

Dynamic data

Dynamic data is the data whose value depends the query term. Score and path are the dynamic data. They need to be stored for every query unlike the static data.
These data will be stored in two different caches. Dynamic data will be stored in a dynamic cache and static data will be stored in a static cache.

**Querying**

Figure 7 represents the querying part in Reduced Data Cache architecture. Query engine first searches the Dynamic cache to get the URI, score and the path of all the resources and then for each URI’s is the result set, static cache is searched to add other metadata like description, URL, preferred label, alternate label and resource type. The combined result will be returned to the user.
Figure 7 Reduced Data Cache
Chapter 7: Scenario Analyses

Performance Analyses

To do the performance analysis, we first realize the performance scenarios by mapping them onto the software architecture and then calculate whether this realization of the scenario meets the requirements set forth in Chapter 4.

Table 2 shows the comparison of query latency between the three architectures.

<table>
<thead>
<tr>
<th>Query Term</th>
<th>Query Results</th>
<th>Query latency in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NoCache</td>
</tr>
<tr>
<td>Immunoglobulin</td>
<td>1003</td>
<td>0.362</td>
</tr>
<tr>
<td>Variable Region</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receptor Platelet</td>
<td>4789</td>
<td>8.484</td>
</tr>
<tr>
<td>Proteolipid Protein</td>
<td>8166</td>
<td>21.974</td>
</tr>
</tbody>
</table>

Table 2 Query latency comparison
Figure 8 shows the Query time taken for the three architectures.

Figure 8 Time taken vs. Total query result
Critique:

- Option 1 has very bad performance for terms with bigger result set. It has good performance for terms with lesser result set.
- Option 2 has the best performance for terms with either bigger or smaller result set.
- Option 3 has better performance than Option 1 but lesser than Option 2.

Cache Memory Analyses

Table 3 shows the comparison of cache memory required between FullDataCache and ReducedDataCache architecture.

<table>
<thead>
<tr>
<th>Query Terms</th>
<th>Memory required in MB</th>
<th>FullDataCache</th>
<th>ReducedDataCache</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td></td>
<td>1817</td>
<td>112</td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td>4500</td>
<td>180</td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td>10500</td>
<td>270</td>
</tr>
<tr>
<td>25000</td>
<td></td>
<td>30000</td>
<td>410</td>
</tr>
</tbody>
</table>

Table 3 Cache Memory comparison
**Critique**

- Option 1 requires no additional cache memory.
- Option 2 requires large amount of memory and growth is exponential.
- Option 3 requires less amount of memory and the growth in linear.

Figure 9 shows the Cache memory required for the two architectures.

![Query Terms vs Memory required](image)

Figure 9 Memory size vs. Query terms
### Table 4 shows the amount time taken to preprocess the results.

<table>
<thead>
<tr>
<th>Total Query terms</th>
<th>Time in minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>28</td>
</tr>
<tr>
<td>5000</td>
<td>90</td>
</tr>
<tr>
<td>10000</td>
<td>166</td>
</tr>
<tr>
<td>25000</td>
<td>442</td>
</tr>
</tbody>
</table>

Table 4 Preprocessing time

**Critique**

- Option 1 does not require any preprocessing of the result.
- Option 2 and Option 3 requires the same amount of time for preprocessing.

Table 4 shows that, as the number of terms increases, preprocessing time increases. Preprocessing time is linearly dependent on total terms. Preprocessing should be done whenever there is an update to triple store. Looking at the time consumed above in the table 2, we can see that preprocessing would cause a lot of delay in updating the cache. To overcome this problem, we can do preprocessing in a distributed manner. Hadoop could be used to distribute the preprocessing over several nodes. Figure 10 shows the architecture of a Hadoop based preprocessing mechanism. There are three nodes in the...
cluster. The query terms will be distributed among the three nodes and preprocessing will happen in the nodes in parallel.

Figure 10 Hadoop based preprocessing
Figure 11 shows the comparison of preprocessing time in a single machine and a five-node Hadoop cluster. We can see almost more than 4 times improvement in a five-node cluster. That is, preprocessing happened 4 times faster in a five-node Hadoop based cluster.
Chapter 8: Critique of the Options

The three architectures can be characterized and understood by the measures that we have just derived. From this analysis, we can conclude the following:

- Option 1 has poor performance. But it has the best affordability in terms of both processing time and memory required.
- Option 2 has excellent performance. But affordability is bad in terms of the memory required.
- Option 3 has better performance than option 1 but less than option 2. It has better affordability than option 1 in terms of memory required.
Chapter 9: Sensitivity Analyses

The performance is correlated positively to the cache memory. That is, we see better performance if we have more cache memory. And affordability is negatively correlated to the cache memory. Affordability will be more when the cache memory is lesser.

We have now discovered the architectural tradeoff point in the cache memory. Performance is correlated positively, while affordability is correlated negatively. It is not possible to maximize performance and affordability simultaneously.
Chapter 10: Conclusion

This research helped in coming up with an architecture, which will improve the performance of querying component of ResearchIQ. This also explores the use of ATAM in architecture evaluation during the early design phase.

Following are the conclusion of the research work:

- ATAM findings were presented to the stakeholders.
- After the discussion, decision was made to use the Reduced Data Caching Architecture.
- Reduced caching improves the query latency on an average by 5 times compared to the basic ResearchIQ querying architecture.
- Reduced caching requires 50 times lesser memory compared to that of Full Data caching Architecture.
- Hadoop based framework for cache population improves the population time by over 4 times using a five-node cluster.
Chapter 11: Future Work

Adding more data sets:

Now that we have an efficient and scalable framework, the focus would be to add more resources to ResearchIQ. There are two future goals in this regard:

1) To integrate more data from the existing data sources. For example, increase the number of publications from PubMed; gather more people profiles from OSUPro, etc. This expansion should be a low hanging fruit to grab, as the end-to-end pipelines for these resources already exist.

2) Integrate new data sources into ResearchIQ. This requires more effort as a new module might have to be written for the data extractor. However, the good news is that the subsequent pipeline is complete.

Adding more data sets will lead to more result set and more queries. We need to consider following things:

1) Data could be dynamic and it may need to be updated every day.

2) Huge amount of data may cause the preprocessing time to increase.
Free Form Search

Currently the user input is restricted to the search terms provided by the auto complete list. The users have to choose a specific term or set of terms from the list else the search cannot be completed. In the future the user should be able to enter free text and the search interface should be able to process the input and find the appropriate concepts. Preprocessing the results in free form search will be very challenging. We need to come up with an architecture, which will support this.

With the free form search, preprocessing the result set will be very challenging. In this architecture presented, we are caching the result set for the restricted search terms. If we allow free form search, then architecture needs to be changed to allow for unrestricted search.

Hadoop based distributed architecture for text annotation component of ResearchIQ

Implementing the MapReduce programming model for the resource annotation pipelines can make data Extraction and semantic annotation pipeline faster. This can be achieved by using Hadoop to distribute the data extraction and annotation among several nodes.
References

Each reference should be referred to in the main body. Make sure the references are complete.


