Data Parallel Application Development and Performance with Azure

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

BY

Yu Zhang
Graduate Program in Computer Science and Engineering
The Ohio State University

2011
Thesis Committee:
Professor. Gagan Agrawal, Advisor
Professor. Feng Qin
ABSTRACT

The Microsoft Windows Azure technology platform provides on-demand, cloud-based computing, where the cloud is a set of interconnected computing resources located in one or more data centers. Recently, Windows Azure platform is available in data centers in most area of the world large cities. MPI is a message passing library standard, which has many participating organizations, including vendors, researchers, software library developers, and users. The goal of the Message Passing Interface is to establish a portable, efficient, and flexible standard for message passing that will be widely used parallel programs. The advantages of developing message passing software using are well known as its portability, efficiency, and flexibility. This thesis is about how to develop MPI like applications on Windows Azure Platform and simulate parallel computing in cloud. The specific goal is to simulate MPI_Reduce and MPI_Allreduce on Windows Azure, and use this simulation to support and build data parallel applications on windows Azure. We also compare the performances of three data parallel applications under three platforms which are traditional clusters, Azure with queue and Azure with WCF.
Dedication

I dedicate this work to my parents Mrs Jiang, Shumin and Professor Zhang, maodi, and my dear advisor Professor Gagan Agrawal
ACKNOWLEDGMENTS

I would like to take this opportunity to thank my adviser, Prof. Gagan Agrawal which is one of the most outstanding professors in the areas of parallel and distributed system and cloud computing. I feel really lucky to have him as my adviser. Professor Gagan Agrawal has been doing research in those areas for decades. His excellent knowledge and foresight is a gold mine to me. With his help and guidance, I finished my thesis step by step and also found out the right direction in computer science I am interested about, which is windows azure and windows communication foundation. I will be remembering and have gratitude to him for the rest of my life. I would also like to thank Dr. Feng Qin for agreeing to serve on my Master’s examination committee.

I would also like to thank Neil Mackenzie---MMVP (Microsoft Most Valuable Professional) and YiYun Luo---MCS (Microsoft Contingent Staff). Both of them work at Microsoft Azure Team, and they have given me a lot of help on some technical details about the mechanism of queue and VMs working behind of Windows Azure.
Vita

2001 to 2005.....................................................B.S. Computer Science, ZhengZhou University
2009 to present....................... ....................M.S Computer Science, Ohio State University

FIELDS OF STUDY

Major Field: Computer Science and Engineering

Studies in:

Data Parallel Application Development in Cloud Environment   Prof. Gagan Agrawal
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>ii</td>
<td>Dedication</td>
<td>iii</td>
</tr>
<tr>
<td>iii</td>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>vi</td>
<td>Vita</td>
<td>vi</td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Emergence of Cloud Computing</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>High Performance Application</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>Studies Ported to Amazon EC2</td>
<td>9</td>
</tr>
<tr>
<td>1.4</td>
<td>Message Passing Interface in Amazon EC2</td>
<td>12</td>
</tr>
<tr>
<td>1.5</td>
<td>Build MPI-like Applications with Azure</td>
<td>13</td>
</tr>
</tbody>
</table>
1.6 Approach Outline ................................................................. 14
1.7 Summery of Work. ............................................................. 15

2. Initial Implementation with Message Queues

2.1 Windows Azure Storage ...................................................... 16
2.2 Algorithms for MPI All Reduce and Reduce with Message .......... 18
2.3 Implement Applications in Azure with Queue. ...................... 23
   2.3.1 Matrix Multiplication. .................................................. 24
   2.3.2 K Means. ................................................................. 24
   2.3.3 KNN. .................................................................. 25
2.4 Summery ................................................................. 25

3. Optimized implementation with WCF

3.1 Introduction to WCF ........................................................... 27
3.2 WCF programming model in Azure. ..................................... 29
3.3 Implement Applications in Azure with WCF ......................... 32
   3.3.1 Matrix Multiplication .................................................... 32
   3.3.2 K Means. ................................................................. 33
   3.3.3 KNN. ................................................................ 34
3.4 Summery ................................................................. 36
4. Experimental Evaluation

   4.1 Experimental Setup .................................................. 38

   4.2 Matrix Multiplication .................................................. 39

   4.3 K Means ............................................................... 41

   4.4 KNN ................................................................. 48

   4.5 Summery ............................................................ 51

5. Conclusions ............................................................ 56

6. References ............................................................. 59
CHAPTER 1
INTRODUCTION

Cloud Computing is a remarkable revolutionary change in information technology and high performance computing that combines computing and data away from personal computers into large Data Centers. Cloud computing is based on a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction[4]. Cloud Computing brings new perspectives in HPC technologies. Developers with innovative ideas for new Internet services no longer require the large capital outlays in hardware to deploy their service or the human expense to operate it[5]. The relevant research has just recently gained momentum and the space of potential ideas and solutions is still far from being widely explored.

The Message Passing Interface (MPI) specification is widely used for solving significant scientific and engineering problems on parallel computers. There exist more than a dozen implementations on computer platforms ranging from IBM supercomputers to clusters of PCs running Windows NT or Linux.

In this thesis, we develop MPI like applications on windows azure platform, and simulate message passing interface reduction and all reduction by using queue and windows communication foundation. The applications we test in this thesis are matrix multiplication; k means classification and k nearest neighbor classification.
1.1 Emergence of Cloud Computing

In the past 30 years, there have been two ways of creating a supercomputer [6]. Firstly, there is the Blue Gene style approach, which creates a massive computer with thousands of CPUs. The other approach, as adopted by Google, is to take hundreds of thousands of small, low cost, computers and hook them together in a “cluster” in such a way that they all work together as one large computer[7]. Cloud Computing has emerged from the existing parallel processing, distributed computing and grid computing technologies. Two cloud computing environments will be mainly talked about in this thesis: Windows Azure and Amazon EC2.

Windows azure, the English word “azure” means a clear blue sky. Because of an advanced promotion and the great effect made by Microsoft, Windows Azure has taken up a considerable part of cloud market. With the increasing construction of datacenters all over the world, windows azure is becoming more and more popular. Windows Azure is the foundation of Microsoft’s Cloud Platform. It is an “Operating System for the Cloud”, which provides essential building blocks for application developers to write scalable and highly available services [9]. Windows Azure provides:

- Virtualized Computation
- Scalable Storage
- Automated Management
- Rich Developer SDK

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud [10]. It is designed to make web-scale computing easier for developers. Amazon EC2’s simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon’s proven computing environment.
Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios. Amazon EC2 provides [11]:

- Select a pre-configured, template image to get up and running immediately.
- Configure security and network access on your Amazon EC2 instance.
- Choose which instance type(s) and operating system
- Pay only for the resources that you actually consume, like instance-hours or data transfer.
- High Performance Application

The Message Passing Interface (MPI) is an industry-standard API specification designed for high performance computing on multiprocessor machines and clusters. The standard was designed jointly by a broad group of computer vendors and software developers, with a number of MPI implementations produced by different research institutes and companies [12]. The most popular one is MPICH, which often is used to optimize MPI implementations for a specific platform. MPI offers a distributed memory programming model for parallel applications. Although the entire MPI API set contains more than 300 routines, many MPI applications can be programmed with less than a dozen basic routines. Figure 1.1 shows four primitives MPI functions: MPI_send(), MPI_recv(), MPI_reduce() and MPI_allreduce(). The first three arguments of both routines specify the location, data type, and size of the message and the fourth argument identifies the target process with which to communicate. The fifth argument provides a further mechanism to distinguish between different messages and the sixth argument specifies the communication context. The receive routine contains an additional argument
to report the message reception status. MPI_REDUCE combines the elements provided in
the input buffer of each process in the group, using the operation op, and returns the
combined value in the output buffer of the process with rank root. The input buffer is
defined by the arguments sendbuf, count and datatype; the output buffer is defined by the
arguments recvbuf, count and datatype; both have the same number of elements, with the
same type. The routine is called by all group members using the same arguments for
count, datatype, op, root and comm. Thus, all processes provide input buffers and output
buffers of the same length, with elements of the same type. Each process can provide one
element, or a sequence of elements, in which case the combine operation is executed
element-wise on each entry of the sequence. MPI includes variants of each of the reduce
operations where the result is returned to all processes in the group. MPI requires that all
processes participating in these operations receive identical results. Same as
MPI_REDUCE except that the result appears in the receive buffer of all the group
members.
1.3. Studies Ported to Amazon EC2

High-performance computing (HPC) uses supercomputers and computer clusters to solve advanced computation problems. The term is most commonly associated with computing used for scientific research. Recently, HPC has come to be applied to business uses of cluster-based supercomputers, such as data warehouses, transaction processing and give rises of the use of cloud computing for cheaper economic solutions.
In this section some commercial HPC applications that have been deployed with clouds has been described by focusing the nature of the application and the commercial benefits of the deployment with the clouds.

1.3.1. The Server Labs (TSL)

The Server Labs (TSL), a European IT company helped their client European Space Agency (ESA) to build the science operations infrastructure for the Gaia project with a goal of making three-dimensional map of our Galaxy by surveying an unprecedented number of stars – more than one billion[13]. The TSL team is helping to build the Astrometry Global Iterative Solution (AGIS) to process all the observations produced by the satellite (1 billion stars x 80 observations x 10 readouts). This requires a tremendous amount of data processing. The team estimated that with the current data set, it would cost approximately 1.5 million Euros for in-house data processing. But the amount of data was increasing each month and each year. They needed a more scalable, cost effective solution. The Server Labs estimated that processing the full 1 billion stars data set with 6 years of data would cost $463,929 on AWS [13]. These calculations helped TSL realize that AWS’s on-demand model would be cheaper and more efficient than buying and maintaining the hardware internally and finally they chose AWS over in house cluster.

1.3.2. Patchwork Diagnostics

Patchwork Diagnostics, a molecular diagnostics company, develops high-value diagnostic tests to aid oncologists in the diagnosis of hard-to-identify cancer tumors. Patchwork chooses optimal models for its tests by using proprietary machine learning algorithms to analyze large libraries of tumor specimen profiles [14]. Unfortunately, the
processing of these models is a highly compute-intensive task. Tens of thousands of models must be processed to find the best model to produce the diagnostic report. While the tests are highly parallelizable, the computation can still take weeks or months using a mid-size high performance computing (HPC) resource, such as a 64-node cluster. Patchwork turned to cloud computing as a solution. They needed access to more computing capacity than they could possibly maintain internally – but only at certain peak times. When they develop and deliver a product for clinical validation they will have weeks where we need access to almost unlimited capacity. The company selected Univac UD’s Unclouded to build HPC clusters in the EC2 cloud. Unclouded is an extension to UniCluster, Univac UD’s product for HPC systems management [14]. With Amazon EC2, Patchwork Diagnostics saved hundreds of thousands of dollars by avoiding the expenses of purchasing and maintaining HPC hardware in-house and were able to accomplish key research innovations which would otherwise have been infeasible.

1.3.3. Cycle Computing

Cycle Computing, a systems integrator specializing in cloud computing and high performance computing, has developed Cycle Cloud to address the growing need to rapidly deploy clusters for pay-as-you-go intensive computing needs [15]. Cycle Computing supported Varian’s implementation of Cycle Cloud for scientific computing needs in the cloud. Varian, Inc. is a leading Scientific Instruments Company, with a reputation for high quality products for scientific industries. Researchers run calculation intensive Monte carol simulations of future products using partial differential equation mathematical models. Varian needed to simulate a design for a mass spectrometer, a very compute intensive operation. The simulations required several thousand compute hours, and nearly 6 calendar weeks on an internal pool of processors. With a conference deadline for completed results looming, they needed to get results done faster. Rather than purchasing hardware, Varian was able to run this several week calculation in under a day using Cycle Cloud. By provisioning a fully secured cluster that grew to several
hundred CPUs on Amazon’s Elastic Compute Cloud (Amazon EC2), Cycle Cloud dynamically scaled up to execute the simulation, then shut down when calculations completed[15]. Cycle Cloud’s pay-per-use model for fully secure, provisioned clusters removes upfront capital expenditures and the costs of an operations team, while enabling access to tremendous compute power.

1.4 Message Passing Interface in Amazon EC2

Amazon Simple Queue Service (Amazon SQS) is a complementary web service that enables you to build highly scalable EC2 applications easily. Amazon SQS is a highly reliable, scalable message queuing service that enables asynchronous message-based communication between distributed components of an application[16]. The components can be computers or EC2 instances or a combination of both. With Amazon SQS you can send any number messages to an Amazon SQS queue at any time from any component. The messages can be retrieved from the same component or a different one right away or at a later time1. No message is ever lost in the interim; each message is persistently stored in highly available, highly reliable queues. Multiple processes can read/write from/to an Amazon SQS queue at the same time without interfering with each other.

![Figure 1.2 Queue Communications in Amazon EC2](image)

1.5. Build MPI-like Applications with Azure
Developing MPI applications in traditional clusters, we should consider that different processors communicate each other by MPI library. The MPI programming model is described as follows:

- Each process in MPI has its own address space.
- Data exchanged between different processes is done by message passing interface.
- The locality issue should be taken into consideration in MPI programming.

However, there is no such library in Azure environment. So in this thesis, the main challenge is to implement message passing between different processes and consider the suitability and feasibility issues about the methods we used for message passing on windows Azure. The first step to build MPI style applications in Azure is to get familiar with the communication mechanism of this cloud computing platform. Just like Amazon EC2, there is message queue in Azure. Besides queue, we also can use WCF as the communication method on Azure. A queue stores messages and makes them available to applications via the REST API [17]. It is highly scalable and available for asynchronous message communication since the messages expiration time can be up to seven days. There can be any number of queues in an account in the Queue service, and there is no limit to the number of messages that can be stored in a queue, but the size of each individual message can’t exceed 8KB. To communicate large object messages, the object can be put in a blob and only its URI will be sent as a message to the queue. Figure 1.3 shows the queue communication in Azure. WCF allows direct communication between role instances. In this approach, roles expose an internal or external endpoint and either bind a WCF service that would listen on this endpoint (if it’s the server) or create a WCF proxy (if it’s the client). The real Uri which should be approached or listened on, should be discovered through the Role Environment.
1.6 Approach Outline of Implementing MPI Reduce and All Reduce

In Windows Azure Platform, assume we have one web role and four worker roles, the web role works as the root processor and each worker role works as individual processor. First we create four queues between the web role and each worker role. Then distribute the workload evenly to each worker role. Each worker role does its own computation and adds the result to its queue. After all the computations have done on each work role, the web role gather the results from each worker role, this operation is done by getting the message from the queues connected with each worker role. Finally the web role processes all the results acquired from queues which works exactly like the MPI reduce operation. For all reduce, the web role can add the result that has been processed to all the queues respectively just like the broadcasting behavior in MPI. All the worker roles then get the result form the queues connected with web role to finish computation. Figure 1.4 shows the inter roles communication model with queues.
1.7 Summary of Work

In chapter 2, we describe how to implement the MPI all reduce and reduce routines in Message Queues and complete three MPI like applications in windows azure with queue. We found the performance of queue communication is fairly slow. So we try the optimized WCF implementation version in chapter 3.

In chapter 3, we explain the three applications implementation with WCF. WCF is a more efficient way in the inter roles communication than message queue. WCF has no data size restriction which in queue is 8k, and WCF is more reliable because it uses the protocol TCP or HTTP which has its embedded security methods. It is a totally different programming model---service oriented the next generation programming model. The performance with WCF is better than queue but which is still far behind the performance of MPI applications in traditional cluster.

In the chapter 4, we test and compare the performances of the three different approaches with two datasets and three applications each.
Chapter 2

Initial Implementation with Message Queues

In this chapter, we first describe the details about message queue, and then explain the algorithms we utilized in Matrix Multiplication, K Means and KNN.

Windows Azure Storage allows application developers to store their data in the cloud, so the application can access its data from anywhere at any time, and store any amount of data and for any length of time, and be confident that the data is durable and will not be lost. Windows Azure Storage provides a rich set of data abstractions:

- Windows Azure Blob – provides storage for large data items.
- Windows Azure Table – provides structured storage for maintaining service state.
- Windows Azure Queue – provides asynchronous work dispatch to enable service communication.

In this thesis, we describe Windows Azure Queue first, and use it to implement MPI recue and all reduce on Azure Environment in chapter 2. As for blob and table, they are only used for storage data instead of communication. Windows Azure Queue allows decoupling of different parts of a cloud application, enabling cloud applications to be easily built with different technologies and easily scale with traffic needs. Windows Azure Queue provides a reliable message delivery mechanism. It provides a simple and asynchronous work dispatch mechanism, which can be used to connect different
components of a cloud application. The Windows Azure Queues are highly available, durable and performance efficient. Its programming semantics ensure that a message must be processed at least once. Furthermore, Windows Azure Queue has a REST(Representational State Transfer) interface, so that applications can be written in any language and they can access the queue at anytime from anywhere in the Internet via the web.

2.1 Queue Data Model

Windows Azure Queue has the following data model.

**Storage Account** – All access to Windows Azure Storage is done through a storage account. This is the highest level of the namespace for accessing queues and their messages. To use Windows Azure Storage a user needs to create a storage account. This is done via the Windows Azure portal web interface. The user will receive a 256-bit secret key once the account is created. This secret key is then used to authenticate user requests to the storage system. Specifically, a HMAC SHA256 signature for the request is created using this secret key. The signature is passed with each request to authenticate the user requests by verifying the HMAC signature. An account can have many queues.

**Messages** – Messages are stored in queues. Each message can be up to 8KB in size. To store larger data, one can store the data in Azure Blob store or Azure Table store, and then store the blob/entity name in the message. Note that when you put a message into the store, the message data can be binary. But when you get the messages back from the store, the response is in XML format, and the message data is returned as base64 encoded. There is no guaranteed return order of the messages from a queue, and a message may be returned more than once.

Definitions of some parameters used by Azure Queue Service are:

- **MessageID**: A GUID value that identifies the message in the queue.
VisibilityTimeout: An integer value that specifies the message's visibility timeout in seconds. The maximum value is 2 hours. The default message visibility timeout is 30 seconds.

PopReceipt: A string which is returned for every message retrieved getting a message. This string, along with the MessageID, is required in order to delete a message from the Queue. This should be treated as opaque, since its format and contents can change in the future.

MessageTTL: This specifies the time-to-live interval for the message, in seconds. The maximum time-to-live allowed is 7 days. If this parameter is omitted, the default time-to-live is 7 days. If a message is not deleted from a queue within its time-to-live, then it will be garbage collected and deleted by the storage system.

2.2 Parallel Application Model

To run a parallel application, a developer should access the Windows Azure portal through Web browser, signing in with windows azure account and create a hosting account for running applications, and a storage account for storing data. Once the developer has a hosting account, he/she can upload his/her applications, specifying how many instances running behind. Windows Azure then creates the necessary VMs and runs the applications.

It’s important to note that a developer can’t supply his/her own VM image for Windows Azure to run [18]. Instead, the platform itself provides and maintains its own copy of Windows. Developers focus solely on creating applications that run on Windows Azure.
The parallel work is done by some number of Worker role instances running simultaneously. Figure 2.1 shows the architecture of parallel application model. Since Windows Azure imposes no limit on how long an instance can run, each one can perform an arbitrary amount of work. To interact with the application, the user relies on a single Web role instance. Through this interface, the user might determine how many Worker instances should run, start and stop those instances, get results, and more. Communication between the Web role instance and the Worker role instances relies on Windows Azure Storage queues or WCF.

2.2 Implementation of MPI Reduce and All Reduce in Azure Queue

1. Create queue in work role by adding the following code in ServiceConfiguration.cscfg file. The connection string name should be set as DataConnectionString.

```xml
<Role name="WorkerRole">
  <Instances count="1" />
  <ConfigurationSettings>
    <Setting name="DataConnectionString" value="UseDevelopmentStorage=true"/>
  </ConfigurationSettings>
</Role>
```
2. Declare the storage account based on the connection string. Create the queue client and finally we create the queue in work role and set the queue reference name to “messagequeue”.

```csharp
// initialize the account information
var storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");
// retrieve a reference to the messages queue
var queueClient = storageAccount.CreateCloudQueueClient();
var queue = queueClient.GetQueueReference("messagequeue");
```

3. Initialize the worker role configuration in OnStart() method

```csharp
CloudStorageAccount.SetConfigurationSettingPublisher((configName, configSetter) =>
{
    configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));
});
```

4. Repeat the above steps in each worker role and make sure all the queues are created and connected with the web role.

The web role pseudo-code of MPI Reduce is as follows

```
While (True)
{
    if (queue1.Exists())
    {
        var msg = queue1.GetMessage();
        if (msg != null)
        {
            DoWork();
            queue1.DeleteMessage(msg);
        }
    }
    if (queue2.Exists())
    {
        var msg = queue2.GetMessage();
        if (msg != null)
        {
            DoWork();
            queue2.DeleteMessage(msg);
        }
    }
    ....
    if (!queue1.Exists()&&(!queue2.Exists()&&(!queue3.Exists()&&……) )
    { Break; }
    Compute();
```

Figure 2.2 Web role MPI Reduce pseudo-codes
As Figure 2.2 pseudo code shows that web role works as the root process and each the worker role works as the individual slave process in MPI environment. Each worker role is connected with web role by queue (as for WCF, each worker role connects with web role by TCP channel, which is shown in next chapter). What MPI reduce function does is to collect all the data from each process and reduce them to the root process. So in windows Azure, Web role first checks all the queues to see if there are any messages exist, then retrieve and store the messages from queues connected with worker roles, and delete the messages immediately after processing. In matrix multiplication, we use this function to reduce all the computation results of final matrix C from worker roles to web role. In K nearest neighbor, web role reduces all distance measures from web roles. As for Kmeans, web role use all reduce function which is shown and explained later in Figure 2.4.

The worker role pseudo-code of MPI Reduce is as follows

```csharp
public class WorkerRole : RoleEntryPoint
{
    public override void Run()
    {
        doWork();
        var msg = new CloudQueueMessage();
        queue.AddMessage(msg);
    }

    public override bool OnStart()
    {
        ReadConfigurationFile();
        Initialize();
        return base.OnStart();
    }
}
```

Figure 2.3 Worker role MPI Reduce pseudo-codes
Worker role MPI Reduce function is shown in Figure 2.3. The worker roles just run simultaneously behind, carry out the computation tasks and add the results into the queues connected with web role.

The web role pseudo-code of MPI All Reduce is as follows

```csharp
While (True)
{
  if (queue1.Exists())
    {
      var msg = queue1.GetMessage();
      if (msg != null)
        {
          DoWork();
          queue1.DeleteMessage(msg);
        }
  }
  if (queue2.Exists())
    {
      var msg = queue2.GetMessage();
      if (msg != null)
        {
          DoWork();
          queue2.DeleteMessage(msg);
        }
  }
  // other...

  if (!queue1.Exists()&&!queue2.Exists()&&!queue3.Exists()&&….)
    {
      Break;
    }
  }
  // get the message to broadcast to all worker roles
  var msg = new CloudQueueMessage();
  queue_1. AddMessage(msg);
  queue_2. AddMessage(msg);
  ..................
```

Figure 2.4 Web role MPI All Reduce pseudo-codes

The web role MPI_Allreduce pseudo-code is shown in figure 2.4. What all reduce function does is reduce then broadcast. Web role gets all the messages form worker roles, and broadcasts the results to all the worker roles after processing. There are 2 queues connected with each worker role and web role. One queue is for sending and the other is for receiving. We use all reduce function in K means. Web role gathers all the mean squared errors from all the worker roles, and performs reduction for global centroids and
global MSE value. New centroids are broadcasted to all the worker roles until no memberships of the points change.

Figure 2.5 shows the pseudo-code of the worker role MPI all reduce. Worker role first checks the receiving queue. If there is a message, worker role then retrieves and stores the message. After processing the message, worker roles then adds the results to the sending queue.

The worker role pseudo-code of MPI All Reduce:

```csharp
public class WorkerRole : RoleEntryPoint
{
    public override void Run()
    {
        if (queue_1.Exists())
        {
            var msg = queue_1.GetMessage();
            if (msg != null)
            {
                DoWork();
                queue_1.DeleteMessage(msg);
            }
            doWork();
            var msg = new CloudQueueMessage();
            queue1.AddMessage(msg);
        }

    public override bool OnStart()
    {
        ReadConfigurationFile();
        Initialize();
        return base.OnStart();
    }
}
```

Figure 2.5 Worker role MPI All Reduce pseudo-codes

One detail should be mentioned here is the project dependency, the web role should depend on all the worker roles, which means we should make sure all the worker roles running before the web role starts. The reason is most of the computation work is done on worker roles, letting worker roles start first can save time on the whole project.
2.3 Implement Applications in Azure with Queue

We have implemented three applications in Windows Azure by queue, Matrix Multiplication, K means and KNN.

2.3.1 Matrix Multiplication

In our first case study, we use the example of matrix-matrix multiplication to illustrate issues arise when developing MPI_like applications in Windows Azure. We have two matrixes A and B, it is straightforward to compute the matrix C= A×B in MPI. We adopt the same algorithm in Azure as the MPI platform.

- Each worker role reads the data from matrix B
- Decouple the matrix A into n parts, n is the number of the worker roles.
- Each worker role gets one part of matrix A, for a N×N matrix, each worker role has two data sets, one is matrix B, the other is part of matrix A, say A_k (1≤k≤n) n is the number of worker roles.
- Each worker role computes the A_k×B and add the result to its queue
- Web role performs the reduce operation gets the final result.

2.3.2 K Means
Parallel $k$-means has been studied previously for very large databases. Parallelization is basically studied for performance advantages mainly the data partitioning and parallel processing. Inherent to classical $k$-means algorithm is intrinsic parallelism. The most intensive calculation to occur is the calculation of distances, and it would require a total of $d \times k$ distance computations where $d$ is the number of documents and $k$ is the number of clusters being created [19]. On the parallel system, the main idea is to split the dataset among processes for faster computation for each partition generated in different processes. Algorithm of Parallel K means in Azure uses the same idea as in Glenn, Where each worker role represents each process in MPI. However it is essential to notice that the communication cost between the worker roles is as considerable as the communication cost in the processes ($T_{\text{communication}} \geq T_{\text{computation}}$)[19]. The communication time is much greater than the computation time. So, for an efficient parallel algorithm, it is necessary to minimize the communication between roles in Azure. The centroid lists consists of $k$ vectors of length $d$ each and in practical applications, it will be much smaller compared to the actual data set, which consists of $n$ vectors each of length $d$. Upon beginning the clustering, each worker role will receive the entire centroid list, but only web role will compute the initial centroids and then broadcast this selected $k$ initial centroids to all worker roles. Each worker role takes care of only the part of the dataset. Thus each worker role is responsible for ($n / \text{number of worker roles}$) vectors rather than the entire set. Each worker role will compute the distances for only part of vectors to each of the centroids, which are all locally stored on each worker role. A series of assignments are generated mapping vectors to clusters. Each worker role then gathers the sum of all vectors allocated to a given cluster and computes the mean of the vectors assigned to a particular cluster. This is repeated for every cluster and a new set of centroids is available on every worker role, and the vectors can be reassigned with respect to the newly calculated centroids.
To compute the quality of the given partition, each worker role can compute the mean squared error for the portion of dataset over which it is operating. As these values are calculated, the worker role will sum its local mean squared error values. A simple reduction of these values among all worker roles will determine the overall quality of the cluster. Figure 2.7 shows the Parallel K means Algorithm.

1. Web role calculates the initial means

2. Broadcast the k centroids to all worker roles

3. Each worker role computes distance of each local document vector to the centroids

4. Assign points to closest centroid and compute local MSE (Mean Squared Error)

5. Perform reduction for global centroids and global MSE value

2.3.3 K Nearest Neighbor

K-Nearest Neighbor or KNN algorithm is part of supervised learning that has been used in many applications including data mining, statistical pattern recognition, and image
processing. The algorithm doesn’t build a classification model but instead it is based on values found in storage or memory. To identify the class of an input, the algorithm chooses the class to which the majority of the input’s k closest neighbors belong to. The KNN algorithm is considered as one of the simplest machine learning algorithms. However it is computationally expensive especially when the size of the training set becomes large which would cause the classification task to become very slow. The parallel implementation greatly increased the speed of the KNN algorithm by reducing its time complexity from $O(D)$, where $D$ is the number of records, to $O(D/p)$ where $p$ is the number of processors[20].

The proposed algorithm is described in the following steps:

1. Web role be the master, the other N-1 worker roles are slaves.
2. Master divides the training samples to N subsets, and distributes 1 subset for each worker role.
3. Each individual worker role now computes the distance measures independently and storing the computes measures in a local array
4. When each worker role terminates distance calculation, it transmits a message to the web role indicating end of processing
5. Web role then notes the end of processing for the sender and acquires the computes measures by reduction.
6. After the web role has claimed all distance measures from all WRs, the following steps are performed:
   a. Sort all distance measures in ascending order
   b. Select top k measures
c. Count the number of classes in the top $k$ measures

d. The input element’s class will belong to the class having the higher count among top $k$ measures

2.4 Case Study

Let’s show the full code example of parallel KNN on Windows Azure with queue. Take 4 processes as example.

1. First, we create one web role and four worker roles in the project.

![Image of role creation]

2. In the property of each worker role, we create a connection string each. These strings are used for the queue communication between each worker role and web role.

![Image of connection string creation]

3. In the property of web role, four connection strings should be declared. The type of the strings should be specified as “Connection String”.

![Image of web role connection string]
4. In worker role OnStart() method, the configuration of queue should be initialized.

```csharp
public override bool OnStart()
{
    // Set the maximum number of concurrent connections
    ServiceManager.DefaultConnectionLimit = 32;
    // For information on handling configuration changes
    CloudStorageAccount.SetConfigurationSettingPublisher(configName, configSetter) =>
    {
        configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));
    },
    return base.OnStart();
}
```

5. In each worker role Run() method, we create a queue for communication with the web role.

```csharp
public override void Run()
{

    // Initialize the account information
    var storageAccount1 = CloudStorageAccount.FromConfigurationSetting("DataConnectionString1");
    // Retrieve a reference to the messages queue
    var queueClient1 = storageAccount1.CreateCloudQueueClient();
    var queue1 = queueClient1.GetQueueReference("messages");
    queue1.CreateIfNotExists();
}
```

6. In web role, four queues are initialized and created. Then the connections between web role and worker roles are ready for use.

```csharp
CloudStorageAccount.SetConfigurationSettingPublisher(configName, configSetter) =>
{
    configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));
};

var storageAccount1 = CloudStorageAccount.FromConfigurationSetting("DataConnectionString1");
var storageAccount2 = CloudStorageAccount.FromConfigurationSetting("DataConnectionString2");
var storageAccount3 = CloudStorageAccount.FromConfigurationSetting("DataConnectionString3");
var storageAccount4 = CloudStorageAccount.FromConfigurationSetting("DataConnectionString4");
var queueClient2 = storageAccount2.CreateCloudQueueClient();
var queueClient3 = storageAccount3.CreateCloudQueueClient();
var queueClient4 = storageAccount4.CreateCloudQueueClient();
```

7. In each worker role, distance measures are computed and added to the queue. This is the reduce part of worker role.

```csharp
StringBuilder MyStringBuilder = new StringBuilder();
for (int i = 0; i < measureCount; i++)
{
    MyStringBuilder.Append(GetValue[1].classLabel);
}

var msg = new CloudQueueMessage(MyStringBuilder.ToString());
queue1.SendMessage(msg);
```
8. In web role, the messages in each queue are retrieved and stored. This is the reduce part in web role.

```java
var queue1 = queueClient1.GetQueueReference("messagequeue");
var queue2 = queueClient2.GetQueueReference("messagequeue");
var queue3 = queueClient3.GetQueueReference("messagequeue");
var queue4 = queueClient4.GetQueueReference("messagequeue");

// retrieve a reference to the messages queue
StringBuilder buf1 = new StringBuilder();
StringBuilder buf2 = new StringBuilder();
StringBuilder buf3 = new StringBuilder();
StringBuilder buf4 = new StringBuilder();

// store messages from the queues
var msg = queue1.GetMessage();
if (msg != null)
{
    buf1.Append(msg.ToString);
    queue1.DeleteMessage(msg);
}

var msg2 = queue2.GetMessage();
if (msg2 != null)
{
    buf2.Append(msg2.ToString);
    queue2.DeleteMessage(msg2);
}

var msg3 = queue3.GetMessage();
if (msg3 != null)
{
    buf3.Append(msg3.ToString);
    queue3.DeleteMessage(msg3);
}

var msg4 = queue4.GetMessage();
if (msg4 != null)
{
    buf4.Append(msg4.ToString);
    queue4.DeleteMessage(msg4);
}
```

After the reduction is finished in web role, web role collects all the distance measures form worker roles, selects the top k measure, sorts in an ascending order, counts the classes in the top k measures and finds out which one has the highest frequency.

2.5 Summary

This chapter introduces Azure message queue data model which is the most essential part of Windows Azure storage data abstraction. We consider the problem of simulating MPI reduce and all reduce with message queue and describe our algorithm which can treat roles as MPI processors to achieve parallelization in Azure environment. Three applications are implemented with Azure queue and we illustrate the algorithm in great details. However there is considerable latency in queue read, write and delete operation.
When the communications between roles are intensive, the performance of queue is intolerable.

So we try WCF as an alternative and optimized solution in next chapter, and compare the pros and cons of these two approaches.
Chapter 3

Optimized Implementation with WCF

In this chapter, we adopt WCF as our inter role communication method as an alternative to queue. Although queue has its many advantages, such as it is a basic and reliable communication way in Azure, and it has its high reliability of message delivery. However, queue has considerable latency for delivering messages. So it is not suitable for parallel computing or high performance computing applications in Azure when the inter role communications are intensive. According to experiments results, we find WCF is a better solution than queue in the implementation of three applications. We have compared the performances in three different scenarios--- traditional clusters, Azure with Queue, and Azure with WCF which are the contents of chapter 4.

3.1 Windows Communication Foundation

Windows Communication Foundation (WCF) is a framework for building service-oriented applications [21]. Using WCF, you can send data as asynchronous messages from one service endpoint to another [22]. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data.

WCF is Microsoft's next-generation programming platform and runtime system for building, configuring and deploying network-distributed services [23]. There are a
number of existing approaches to building distributed applications. These include Web services, .NET Remoting, Message Queuing (MSMQ) and COM+/Enterprise Services. Figure 3.1 shows all the technologies combined to form WCF. WCF provides a common platform for all .NET communication. It unifies these into a single framework for building and consuming services [24]. Instead of building business logic as a set of classes in an assembly, WCF build them as a set of independent services that we make available as a service. WCF services expose functionality through one or more endpoints. Clients communicate with the service's endpoint.

In configuring an end points there are three components require. Address, Binding and Contract [25]. These three components are interrelated. An address means an URL. Binding tells the WCF runtime how your endpoint communicates. Contracts mean it gives the information of working endpoints. Figure 3.2 shows the architecture of WCF and its three main components.

With the help of XML-based configuration information we can configure an end point, programmatically using methods/properties, or with both XML configuration files and some coding [26].
3.2 WCF Programming Model in Azure

The main idea of WCF is that each worker role hosts a service which can be called by web role to do the computation and return the results to web role by TCP channels.

1. Define Service Contract in each work role.

Service Contracts describe which operations the client can perform on the service. There are two types of Service Contracts. Figure 3.3 shows the Pseudo-code WCF Service Contract Declaration.

[ServiceContract] - This attribute is used to define the Interface.

[OperationContract] - This attribute is used to define the method inside Interface.

Any services built using a contract of this type expose no operations for clients to use. This topic describes the following decision points when designing a service contract:

- Whether to use classes or interfaces.
- How to specify the data types you want to exchange.
- The types of exchange patterns you can use.
- Whether you can make explicit security requirements part of the contract.
2. Define WCF Endpoint

The Endpoint's Address is a network address where the Endpoint resides.

The Endpoint's Binding specifies how the Endpoint communicates with the world including things like transport protocol (e.g., TCP, HTTP), encoding (e.g., text, binary), and security requirements (e.g., SSL, SOAP message security).

The Endpoint's Contract specifies what the Endpoint communicates and is essentially a collection of messages organized in operations that have basic Message Exchange Patterns (MEPs) such as one-way, duplex, and request/reply.

Adding the following code to ServiceDefination.cscfg file, we thus create an endpoint in role. We also can specify the protocol to be HTTP and assign a specific port to this endpoint. Otherwise the port number will be dynamically assigned.

```
[ServiceContract]
public interface IMM
{
    [OperationContract]
    string compute();
}
```

```
<Endpoints>
    <InternalEndpoint
        name="Endpoint"
        protocol="tcp"/>
</Endpoints>
```

WCF services are deployed, discovered, and consumed as endpoints. Endpoint is composed of three components: Address, binding and contract. Address shows where the service is. Binding specifies what protocol should be used. Contract illustrates what service we can use.
An address uniquely identifies a service provides the transport protocol, name of target machine and port. Binding provides a canned method regarding protocol, message encoding, communication pattern, reliability, security policies and so on and so forth. There are 5 types of contracts. We only use 3 of them in the applications. The xml code below shows how to declare an endpoint in configuration file. In the service tag, address shows the URI of the service, binging shows the protocol we use is TCP and the contract illustrates the service type is IMM.

```xml
<!-- configuration file used by above code -->
<configuration
 xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
 <system.serviceModel>
 <services>
  <!-- service element references the service type -->
  <service type="MM">
   <!-- endpoint element defines the ABC's of the endpoint -->
   <endpoint
      address="http://localhost/MM/Ep1"
      binding="netTCPBinding"
      contract="IMM"/>
  </service>
 </services>
 </system.serviceModel>
</configuration>
```

3. Host Service in Worker Role.

After defining the endpoint in worker role, the next step is to start the service. There are two ways adding the endpoint to the service host. Figure.3.4 shows the pseudo-code.
The above code in figure 3.4 shows two ways of hosting a service. In the first part, we declare ServiceHost, name it “sh”, and add endpoint to the host. Finally we open this service host and wait for call from web role. The second part shows how to declare the endpoint in configuration file, which has the same functionality as the code we implement in workerrole.cs file.

4. Create Channel Factory and Call the Service

The WCF channel model represents the main entry point for implementing new transports, message encoding mechanisms or protocols not supported out of the box in the WCF infrastructure. The three main components of the channel model are protocols,
encodings and transports, and any of these can be customized or extended to support new functionality. These three main components are used to create a communication stack, also called a channel stack, which is mainly composed of different abstraction layers or channels. As any architecture based on layers, each channel in the stack provides some abstraction about the implementation of a protocol or transport, and exposes this abstraction to the channel directly below it. Messages flow through this communication stack as Message objects, and depending on the functionality of each channel, they can modify or include metadata in the messages, provide a message encoder to transform the messages into a stream of bytes, or simply send or receive the messages through a transport. Figure 3.5 shows the architecture of WCF Channel Layer. The channel works like the queue, but doesn’t have any limitations like queue does.

The follow pseudo code shows how to create a channel in web role and call the service hosted on work role. In web role, we simply create a TCP channel factory, call the service provided in worker role.

```csharp
NetTcpBinding b = new NetTcpBinding(SecurityMode.None);
var factory = new ChannelFactory<WorkerRole.IMM>(b);
var channel = factory.CreateChannel(GetEndpoint());
channel.compute(); // call the service hosted on work role
```

5. Implement MPI_Reduce and MPI_Allreduce with WCF
Once the channel is created, it works just like the queue, but doesn’t have any limitations like queue does. This channel between the worker role and web roles is a TCP connection, and the buffer size can reach up to 2 GB, unlike the 8k limitation in queue. Message gathering and broadcasting as well as the reduction and all reduction is done by channels. The algorithm of reduction and all reduction is exactly same as it is in queue.

3.3 Implement Applications in Azure with WCF

3.3.1 Matrix Multiplication

As we mentioned in chapter 2, the message size in queue is limited under 8k. But for WCF, there is no such limitation. So we don’t have to care about the message size as opposed to what we did in queue. For a 1000 by 1000 integer matrix, each column or each row has 1000 integer elements. Each integer takes 8 bytes in memory space, so one row or one column of the matrix takes 8k which is the max message size can be transferred by queue. Each worker role should add the message to queue before the message size reaches its max value. Otherwise the message cannot be recognized and processed by queue. In the scenario of WCF version of Matrix Multiplication, we send message from worker roles to web role after all the computations are done in worker roles instead of sending message less or equal than 8k in queue scenario. But we still need to be aware of some WCF properties such as maxReceivedMessageSize and maxBufferSize, the default value of maxReceivedMessageSize and maxBufferSizes is set to 65535, the default value should be increased by modifying the correspondent configuration properties. Otherwise you will be getting an error message like: “The maximum message size quota for incoming messages (65536) has been exceeded…” The following code shows how to change the settings. It changes the maximum of received message size to 10MB, actually the maximum message size can reach up to 2147483647 bytes, which is 2GB. But we better not set the maximum message size way
to large which may cause some network security issues. What’s more, in the case of retransferring a lost or corrupted message, it takes a long time to recover from failure. You may notice the reliableSession attribute is enabled. This is related to two main options for transferring message in WCF: streamed or buffered. Reliable messaging cannot be used with streaming as the TCP mechanism requires processing the data as a unity to apply checksums, etc. Processing a message this way would require a huge buffer and thus a lot of available memory on both client and server; but it may cause the denial-of-service issues. We adopt buffered transformation for all the applications in WCF: splitting the whole message into 64KB fragments and using reliable, ordered messaging to transfer the data.

```xml
<binding name="NetTcpBinding"
  transactionFlow="true"
  maxBufferSize="10485760"
  maxReceivedMessageSize="10485760">
  <reliableSession enabled="true" />
  <security mode="Transport">
    <transport clientCredentialType="Windows" protectionLevel="EncryptAndSign" />
  </security>
</binding>
```

Basically there are two major parts in WCF applications we have done in my thesis: 1. worker roles host the service, 2. Web role call the service. Web role also can host IIS service open to external service requests from outside clients who are using a different platform or diverse programming languages, then using the load balancer to distribute workload to internal web roles. In this thesis, all the computations are done in the internal roles, so we don’t utilize IIS service in web role.

1. Matrix Multiplication Algorithm in WCF worker role

Worker roles define the service in ServiceContract in terms of a C# interface IMM(), and implement the interface in the OperationContract MM(). The algorithm we use here
are identical to the one we use in queue, the only difference is we do not care about how the messages are being transferred between the roles, and WCF takes care of message passing based on the endpoint we define in the configuration file.

The following code shows the service class MM running on the worker role.

```csharp
namespace WorkerRole
{
    public class MM : IMM
    {
        public string compute()
        {
            DoWork(); // multiply Matrix B and the subset of Matrix A
            return result;
        }
    }
}

NetTcpBinding b = new NetTcpBinding(SecurityMode.None);
var factory = new ChannelFactory<WorkerRole.IMM>(b);
var endpoints = RoleEnvironment.Roles[].Instances.Select(i => i.InstanceEndpoints[]).ToArray();
var ep1 = endpoints.IPEndPoint; // URI is the address of the hosted service
var channel = factory.CreateChannel(EndpointAddress(String.Format("net.tcp://{0}/MM", endpoints[].IPEndPoint).
channel.compute(); // call the service hosted on work role
```

2. Matrix Multiplication Algorithm in WCF web role

Web role creates the channel between itself and all the worker roles based on the endpoint defined in worker role, and call the service hosted in worker roles.

3.3.2 K Means

Service hosted in WCF K means is slightly different from matrix multiply and k nearest neighbor. K means requires a duplex communication between the web role and worker roles. While Matrix and KNN just utilize one way communication.
Figure 17 shows the duplex communication in WCF.

![Diagram of duplex communication in WCF](image-url)

When running duplex communication in WCF using the netTcpBinding - the WCF runtime automatically sets up a callback address for the calling application to listen on based on a value provided in a binding configuration. This is a unique address, unique per worker role. If you want to run several client applications, it is necessary to have a really unique client-side callback address per instance. Following is an app.config for the client together with a code snippet to set up application instance-specific callback addresses in WCF version of K means.

```xml
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="Duplex"
        address="http://localhost/Kmeans"
        binding="netTcpBinding"
        bindingConfiguration="customizedCallbackBinding"
        contract="IKmeans" />
    </client>
    <bindings>
      <netTcpBinding>
        <binding name="customizedCallbackBinding"
          clientBaseAddress="http://localhost:7777/kmeans" />
      </netTcpBinding>
    </bindings>
  </system.serviceModel>
</configuration>

netTcpBinding = (netTcpBinding)proxy.Endpoint.Binding;
string clientCallbackAddress = binding.ClientBaseAddress.AbsoluteUri;
clientCallbackAddress += Guid.NewGuid().ToString();
binding.ClientBaseAddress = new Uri(clientCallbackAddress);
```

38
3.3.3 K Nearest Neighbor

The pattern of WCF version of K nearest neighbor is similar to WCF Matrix Multiplication. One way communication, services are hosted in worker roles and web role calls the service to do the computation. The algorithm we adopt in WCF KNN is identical to Queue KNN. The protocol we use in WCF KNN is also TCP. TCP can scale and behave better than HTTP in most WCF scenarios. The duplex for TCP is implemented at the network layer rather than the application layer. WsDualHttp requires WS-ReliableMessaging and a client-side service to implement duplex. It only supports security at the message level while TCP supports security at either the message or transport level. The following is the pseudo code for WCF version of KNN

Worker roles host service--- KNN

```csharp
namespace WorkerRole
{
    public class KNN : IKNN
    {
        public string compute(string path)
        {
            DoWork(); // Get the K nearest neighbor for each test sample
            return result;
        }
    }
}
```

```csharp
NetTcpBinding b = new NetTcpBinding(SecurityMode.None);
var factory = new ChannelFactory<WorkerRole.IKNN>(b);
var endpoints = RoleEnvironment.Roles[0].Instances.Select(i => i.InstanceEndpoints[]).ToArray();
var ep1 = endpoints.IPEndPoint; // URI is the address of the hosted service
var channel = factory.CreateChannel(EndpointAddressFactory.Create("net.tcp://{0}/KNN".), endpoints[0].IPEndPoint());
channel.compute(path); // call the service hosted on work role and get result
```

Web role calls the service through TCP channel
3.5 Case Study

Let’s look at the implementation of KNN with WCF in full code example.

1. Create 1 web role and 4 worker roles

2. In the property of each worker role, add endpoint and specify the type, and protocol. The port number is dynamic assigned.

3. In the property of web role, add endpoint to accept the outside request. Its type is input and protocol is HTTP.

4. Declare service contract and operation contract in worker role. The interface IKNN is implemented in the compute() method which is the content of operation contract.
5. Initialize the endpoint in worker role `Onstart()` method.

```csharp
public override bool Onstart()
{
    var baseAddress = new Uri("net.tcp://(local)\{Endpoint1\}");
    host = new ServiceHost(typeof(INN), baseAddress);
    host.AddServiceEndpoint(new IPEndPoint(IPAddress.Any, 12345),
                            new NetTcpBinding(), "INN");
    host.Open();
    return base.Onstart();
}
```

6. Create channels in web role and call the service provided in worker role.

```csharp
NetTcpBinding b1 = new NetTcpBinding(SecurityMode.None);
b1.MaxBufferSize = 2147483647;
b1.MaxReceivedMessageSize = 2147483647;
DistributedTransactionManager trans = new DistributedTransactionManager();
var channel2 = new ChannelFactory<WorkerRole1>(b2);
string a = channel2.CreateChannel("Endpoint1");
string a = channel1.CreateChannel("Endpoint1");

NetTcpBinding b2 = new NetTcpBinding(SecurityMode.None);
var channel2 = new ChannelFactory<WorkerRole2>(b2);
string b = channel2.CreateChannel("Endpoint2");

NetTcpBinding b3 = new NetTcpBinding(SecurityMode.None);
var channel3 = new ChannelFactory<WebRole3>(b3);
string c = channel3.CreateChannel("Endpoint3");

NetTcpBinding b4 = new NetTcpBinding(SecurityMode.None);
var channel4 = new ChannelFactory<WorkerRole4>(b4);
string d = channel4.CreateChannel("Endpoint4");
```

Once the TCP channels are created between worker role and web role, message passing, including gathering and broadcasting is done via the channels.

### 3.6 Summary
The chapter considered the WCF implementation of the three MPI applications. We introduce the WCF programming model which is service oriented, and describe the WCF mechanism in Azure environment. The problems of WCF message size and buffer are also being considered. We describe how the three applications are implemented in WCF approach. Comparing to queue, WCF is more efficient, basically no latency, more reliable, TCP protocol can ensure the safety of message in the transportation layer, but much more complicated, there are so many contracts we should specify and take into consideration.
Chapter 4

Experimental Evaluation

This chapter describes the experiments conducted to evaluate our implementations in three scenarios, traditional clusters, Windows Azure with queue and WCF. Two data sets have been used in each implementation. The goals of our experiments are designed to compare the performances of MPI like applications in the three approaches, and to test the performance of Azure message queue and WCF in the inter roles communication. The programming language we use in MPI is C and C++, while in Windows Azure is C#.

4.1 Experimental Setup

We choose Glenn, the IBM Opteron Cluster in Ohio Super Computing Center, Linux operating system. The 88 System x3755 compute nodes support multi-core servers, comprising Quad socket, dual core Opterons and 64 GB (2 nodes), 32 GB (16 nodes), or 16 GB (70 nodes) RAM. Each core has a frequency of 2.6GHz.

Windows Azure cloud operating system works as an online service, with automated provisioning and service management. It has multiple roles and each role instance runs on its own virtual machine. Figure 4.1 shows options that are available for each virtual machine size. The frequency of each core in Azure is 1.6 GHz; with 1.75 GB RAM, and 100 MB network bandwidth.
4.2 Performance Comparison

We measure the execution time in two aspects, computation time and communication time. The computation time in traditional cluster is straightforward, and it is closely related to the floating point operations per second of the CPU. The communication time in traditional cluster is also simple, and it is determined by the message passing interface.

In windows Azure, application execution time is affected by numerous factors, such as objects initialization, compiler issues, performance of queue, data serialization, network quality, congestion problems in TCP and so on. The C# compiler does not have as many sophisticated optimizations as does the C or C++ compiler. All code is .net framework should be packaged in assemblies. All these factors make Azure much slower than traditional clusters. We also measure the execution time in Windows Azure in two aspects, communication time and computation time. Most of the computation times take place in worker roles, and communication time is determined by the performance of queue and TCP channel in WCF. But the performance of queue varies greatly, and TCP channel is also not stable compared to MPI. We measure the performance time in average.

4.2.1 Matrix Multiplication
We start with matrix multiplication with small dataset using two 500 ×500 integer matrixes. The algorithms we use are the same in three different scenarios. In figure 4.2 we can see, the performance Glenn outweighs the other two to a great extend. WCF is better than queue because there are no queue communication occurrences in WCF. With
the increasing worker roles, the computation times in windows azure decrease, but extra money has to pay for the additional worker roles running in windows azure.

![Figure 4.3 Communication Time of Matrix Multiplication large Dataset in Three Platforms with multiple processors](image)

**Table 4.3 Computation Time of Matrix Multiplication in Large Dataset**

<table>
<thead>
<tr>
<th>Processors</th>
<th>MPI</th>
<th>Queue</th>
<th>WCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Processors</td>
<td>0.4633sec</td>
<td>29.9872sec</td>
<td>18.0985sec</td>
</tr>
<tr>
<td>4 Processors</td>
<td>0.7032sec</td>
<td>63.2409sec</td>
<td>34.8856sec</td>
</tr>
<tr>
<td>2 Processors</td>
<td>1.1092sec</td>
<td>109.9873sec</td>
<td>69.1129sec</td>
</tr>
</tbody>
</table>

**Table 4.4 Communication Time of Matrix Multiplication in Large Dataset**

<table>
<thead>
<tr>
<th>Processors</th>
<th>MPI</th>
<th>Queue</th>
<th>WCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Processors</td>
<td>1.0266sec</td>
<td>4.2330sec</td>
<td>4.2352sec</td>
</tr>
<tr>
<td>4 Processors</td>
<td>2.3235sec</td>
<td>7.0990sec</td>
<td>7.0985sec</td>
</tr>
<tr>
<td>2 Processors</td>
<td>3.3745sec</td>
<td>11.0980sec</td>
<td>13.0981sec</td>
</tr>
</tbody>
</table>
In the second experiments in Matrix Multiplication, we apply two 1000×1000 integer matrixes as our large dataset. It’s a both data and communication intensive application.

We add the computation result of each row in final Matrix C directly into queue, so there are at least 1000 queue iterations totally in this application. So there is significant time consumption in queue. For 2 processors in queue, the time usage even reaches up to almost 2 minutes.

4.2.2 KMeans

![Figure 4.4 Communication Time of KMeans Small Dataset in 3 Platforms with multiple processors](image)

<table>
<thead>
<tr>
<th>Processors</th>
<th>MPI</th>
<th>Queue</th>
<th>WCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 processors</td>
<td>0.6423 sec</td>
<td>1.8902 sec</td>
<td>1.9234 sec</td>
</tr>
<tr>
<td>4 Processors</td>
<td>0.9512 sec</td>
<td>2.1224 sec</td>
<td>2.0267 sec</td>
</tr>
<tr>
<td>2 processors</td>
<td>1.4623 sec</td>
<td>3.6238 sec</td>
<td>3.5263 sec</td>
</tr>
</tbody>
</table>

Table 4.5 Computation Time of KMeans in Small Dataset
We use Iris data set as our experimental sample in K Means. The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by Sir Ronald Aylmer Fisher (1936) as an example of discriminate analysis [27]. Four features were measured from each sample, and they are the length and the width of sepal and petal, in centimeters. Based on the combination of the four features, we classify the 1 million data sets into 3 subsets in our experiment. Where $k$ equals to 3, there are setosa, versicolor, and virginica respectively in real world.

<table>
<thead>
<tr>
<th>MPI</th>
<th>Queue</th>
<th>WCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Processors</td>
<td>0.1743 sec</td>
<td>2.1902 sec</td>
</tr>
<tr>
<td>4 Processors</td>
<td>0.2612 sec</td>
<td>3.9344 sec</td>
</tr>
<tr>
<td>2 processors</td>
<td>0.6043 sec</td>
<td>7.0231 sec</td>
</tr>
</tbody>
</table>

Table.4.6 Communication Time of KMeans in Small Dataset

![Figure 4.5 K Means Large Dataset in 3 Platforms with multiple processors](image)

<table>
<thead>
<tr>
<th>MPI</th>
<th>Queue</th>
<th>WCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Processors</td>
<td>1.0233 sec</td>
<td>4.9878 sec</td>
</tr>
<tr>
<td>4 Processors</td>
<td>1.9374 sec</td>
<td>6.8988 sec</td>
</tr>
<tr>
<td>2 processors</td>
<td>2.6023 sec</td>
<td>9.2823 sec</td>
</tr>
</tbody>
</table>

Table.4.7 Computation Time of KMeans in Large Dataset
The large data set for Kmeans is 5 million, and k is still 3. As we can see, the performance in MPI is still the best. For queue and WCF, the communication between inter roles in Kmeans is much less than that in matrix multiplication. So the computation time is tolerable compared to matrix multiplication. The communication in Kmeans between the worker roles and web role is duplex, both worker role and web role need add and retrieve message to and from queues, but the total queue iterations are under 100. For WCF, messages are sent directly based the URI addresses of roles, the TCP channel is almost twice faster than queue.

<table>
<thead>
<tr>
<th></th>
<th>MPI</th>
<th>Queue</th>
<th>WCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Processors</td>
<td>0.2872 sec</td>
<td>2.0623 sec</td>
<td>1.8976 sec</td>
</tr>
<tr>
<td>4 Processors</td>
<td>0.7567 sec</td>
<td>4.3098 sec</td>
<td>2.3214 sec</td>
</tr>
<tr>
<td>2 processors</td>
<td>1.2233 sec</td>
<td>7.2356 sec</td>
<td>4.1218 sec</td>
</tr>
</tbody>
</table>

Table 4.8 Communication Time of KMeans in Large Dataset

4.2.3 K Nearest Neighbor

![Figure 4.6 Communication Time of KNN Small DataSet in 3 Platforms with multiple processors](image)
The data set we use in KNN is iris flower as well. 1 million samples are selected as the training data set and 100 samples are the testing data set. K value is set to 7. We notice that the performances of KNN in three scenarios have slight difference compared to matrix multiplication and k means. Actually, the roles communication in KNN only occurs once, which is each worker role sends its own seven nearest neighbors with top seven frequency lists to web role, and then web role calculates the final results. The communication is only one way from worker roles to web role.
For large data set in KNN, we choose 5 million samples as the training data set and 100 samples as the testing data set. K value is still set to 7. As we can see, WCF is always faster than Queue. The computation time of queue varies greatly. The average read and writes time of queue is about 60ms, but in the worst case scenario, the IO time of queue can reaches up to 600ms.

Table 4.11 Computation Time of KNN in Large Dataset

<table>
<thead>
<tr>
<th></th>
<th>MPI</th>
<th>Queue</th>
<th>WCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Processors</td>
<td>1.3089 sec</td>
<td>3.9978 sec</td>
<td>3.0087 sec</td>
</tr>
<tr>
<td>4 Processors</td>
<td>2.5619 sec</td>
<td>7.3001 sec</td>
<td>8.3420 sec</td>
</tr>
<tr>
<td>2 Processors</td>
<td>4.0329 sec</td>
<td>12.9962 sec</td>
<td>11.9891 sec</td>
</tr>
</tbody>
</table>

Table 4.12 Communication Time of KNN in Large Dataset

<table>
<thead>
<tr>
<th></th>
<th>MPI</th>
<th>Queue</th>
<th>WCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Processors</td>
<td>0.0889 sec</td>
<td>1.9978 sec</td>
<td>1.0087 sec</td>
</tr>
<tr>
<td>4 Processors</td>
<td>0.1019 sec</td>
<td>4.3001 sec</td>
<td>3.3426 sec</td>
</tr>
<tr>
<td>2 Processors</td>
<td>0.3329 sec</td>
<td>7.9962 sec</td>
<td>6.9891 sec</td>
</tr>
</tbody>
</table>

For large data set in KNN, we choose 5 million samples as the training data set and 100 samples as the testing data set. K value is still set to 7. As we can see, WCF is always faster than Queue. The computation time of queue varies greatly. The average read and writes time of queue is about 60ms, but in the worst case scenario, the IO time of queue can reaches up to 600ms.
4.5 Summary

Based on the experimental evaluation, we notice Glenn is much faster than Windows Azure in data intensive applications. This performance difference can be explained and analyzed in several aspects. First of all, the goal of Windows Azure is not designed to achieve high performance computing. We investigate the reasons in the following aspects.

1. Hardware

Glenn has a higher CPU frequency and larger RAM size. Glenn has an average CPU frequency up to 2.6 GHz, but for Azure, it only has 1.6 GHz. With respect to RAM size, Azure is incomparable to Glenn as well.

2. Operation System

The OS running on Glenn is Linux which has a lightweight kernel can make full use of hardware resources.

3. Programming Language

C is only one level of abstraction away from machine language. C# running on the .Net framework is at a minimum 3 levels of abstraction away from assembler.
5 Conclusions

To explore the relationship between Windows Azure Platform and Message Passing Interface Standard in terms of migrating MPI Style Applications to Windows Azure is a promising and practical topic. It has treble meanings.

First, MPI applications can harness the advantages of cloud computing. Windows Azure is an online form of computing where users can access applications via a browser, while the application is installed and stored on a server as well as the data. This is a whole new form of computing and is allowing thousands of users from all around the world to access something without having to download and install on their own computers. The Windows Azure platform abstracts hardware resources through virtualization. Each application that is deployed to Windows Azure runs on one or more Virtual Machines (VMs). These deployed applications behave as though they were on a dedicated computer, although they might share physical resources such as disk space, network I/O, or CPU cores with other VMs on the same physical host. So than we can even run some MPI applications in our Smartphone which has built-in web browser, and access applications and data just through web browser without installing any client applications.
Second, Applications running on the cloud can achieve high efficiency by simulation of MPI parallelization on Windows Azure Platform. We distribute the total workload evenly to several worker roles in a controllable way. MPI includes variants of each of the reduce operations where the result is returned to all processes in the group. MPI requires that all processes participating in these operations receive identical results. So in Windows Azure, each worker role is responsible for part of total computation, then using the notation of MPI reduce and all reduce to communicate and exchange the results on different roles, returning the desired results to web role or broadcasting the result to all worker roles from web role to achieve high efficiency.

Third, we introduce the different inter roles communication methods in parallel way which can be considered as a prototype of Azure MPI Library which most likely will be developed and unitized in the near future. We test the performance of the different approaches. One is queue and the other is windows communication foundation, and find out WCF is more appropriated and efficient compared to queue in the internal roles communication.
References


Page 53-65.


