TCP/IP Implementation of Hadoop Acceleration

by

Cong Xu

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
Aug 4, 2012

Keywords: Cloud Computing, Hadoop, TCP/IP

Approved by

Weikuan Yu, Chair, Assistant Professor of Computer Science and Software Engineering
Dean Hendrix, Associate Professor of Computer Science and Software Engineering
Saad Biaz, Associate Professor of Computer Science and Software Engineering

Abstract

Cloud Computing is a booming technology in computer science. Since Google released the design details of the MapReduce technique in 2004 [1], cloud computing has been more and more popular. Hadoop [2] has been developed as an open-source implementation of MapReduce.

A new network-levitated merge mechanism (Hadoop-A) [3] improves the existing Hadoop framework to solve many problems in the original framework. Hadoop-A avoids repetitive merging of data and introduces a full pipeline that consists of shuffle, merge and reduce phases. However, Hadoop-A is implemented based on Infiniband RDMA technology, which is not commonly deployed on commercial servers. On the other hand, data transmission based on the TCP/IP protocol is a robust technology, its speed is becoming faster and faster. Thus, we deem that it worthwhile to complement our RDMA-based connection with an implementation that is built on TCP/IP protocol.

In this article, I will describe the details of design and implementation of a TCP/IP implementation of Hadoop-A. Two components MOFSupplier (Server) and NetMerger (Client) are introduced to realize the TCP/IP connection, which can fetch data from Maptasks and send them to Reducetasks within the new network-levitated merge mechanism. Multithreading technologies are used to manage memory pool, send/receive and merge data segments. The experiment results show that the TCP/IP implementation can bring good performance for Hadoop-A on TCP/IP. Its execution time outperforms original Hadoop by 26.7% and can also achieve good scalability.

Acknowledgments

This research project would not have made progress without the suport of a lot of people. Firstly, I'd like to express my special gratitude to my advisor, Dr. Weikuan Yu, for his kind guidance, encouragement and patience in achieving the goal of this project. I have to say, Dr. Yu has made available his support in a number of ways. He not only gives instruction on my research project but also continues to encourage my spirit, especially when I got involved in some terrible troubles. I really want to express my sincere appreciation for his help. It is really an honor for me to be his student.

I also want to thank my advisory committee members, Dr. Dean Hendrix, Dr. Saad Biaz and all of the professors and staff members in Computer Science and Software Engineering Department who have kindly helped me in these three years. I am indebted to my colleagues in the Parallel Architecture and System Laboratory: Yandong Wang, Patrick Carpenter, Zhuo Liu, Xinyu Que, Bin Wang and Yuan Tian, who have helped me a lot on my research work. They always make me feel that we are in one family. I can never feel lonely.

Finally, I would like to express my sincere thanks to my parents, especially my father. I always have him behind me and there's a lot of love, a lot of support and it's just great.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing has been popular for several years, it has been ranked as one of the most leading emerging technologies since 2008. Gartner Inc. has predicted that 80 percent of Fortune 1000 companies would pay for cloud-computing service, and 30 percent of these companies can pay for cloud-computing infrastructure [4].

Cloud Computing is defined to be the provision of computational resources on demand via a computer network, such as applications, databases, file services, emails, etc. [5] A great many companies have invested a large amount of money and time on cloud computing, for the reason that it can use lots of resources in an effective way. Lots of big companies including Google, Yahoo, Facebook and IBM have made great contributions to cloud computing technology and achieved remarkable successes both commercially and technically.

Customers can reap many benefits from cloud service. They can cut down on their capital expenditures and take advantage of operational expenditures to increase computing capabilities, which requires fewer IT support personnel. Companies can also change deployment size to match requirements very quickly; as a result, the flexibility of cloud services enables their customers to use more resources at peak times. It is also convenient to access cloud services everywhere. With the help of multiple redundant sites, services are more reliable, and it is easy to achieve the goal of disaster recovery and business continuity. At the same time, the cost of maintenance can be reduced.

Cloud computing can provide not only broad commercial opportunities for big IT companies but also huge research space for high-performance computing scientists. By now, a large number of new technologies have been deployed in the cloud.

In cloud computing, Map Reduce is a programming model for processing and analyzing large data sets. Users can first create a function handling a Map based on key/value pair collection of data; then create a Reduce function to combine all of the intermediate key values with the same value of the intermediate value.

The framework of the Map Reduce program can be executed in parallel on a large number of computers, and this system focuses on how to split the input data, schedule the execution of programs on a large number of machines, deal with machine failures, and manage the communication between computers.

Google File System (GFS) [6] and Hadoop Distributed File System (HDFS) [7] are two file systems to support Map Reduce framework. In order to satisfy the fast growing demands of data processing requirements, Google File System (GFS) has been developed, which is a scalable distributed file system for large distributed data-intensive applications. It also supports the feature of fault tolerance when deployed on normal commercial cluster, and can achieve very high aggregated I/O performance. The Hadoop Distributed File System(HDFS) is used to reliably store large files across different nodes in the cluster. In the Hadoop Distributed File System, files are stored as a sequence of blocks. The sizes of the blocks, except the last one, are the same. In a file, blocks are replicated for fault tolerance. Files in HDFS are "write once" and only one writer is allowed at any time.

Hadoop is an open source implementation of MapReduce. It is a new way for companies to store and process data [8]. It contains two key components: data storage mechanism with the help of Hadoop Distributed File System and high-performance large-scale data processing using MapReduce framework.

Hadoop has the ability to run on large scale commercial, shared-nothing servers. It is easy to add or remove servers in a cluster running Hadoop and it can automatically detect and recover from system or hardware failure.

On top of Hadoop program, Apache Pig [9] and Hive [10] are supported for data processing and analizing, which are two examples using Hadoop. Pig is a high level execution

framework for parallel computing which are used to deal with large amount of data. The Hive data warehouse software helps to manage and query big data sets stored in distributed file system.

InfiniBand is a standard switched fabric adopted in high-performance computing and industry data centers. It has a lot of benefits such as: high throughput, low latency and good scalability. Remote Direct Memory Access(RDMA) is supported by InfiniBand. RDMA has the capability of directly accessing remote computers' memory without the operating system being involved. RDMA achieves zero-copy communication to exchange data to or from application memory, reducing the requirement of data copying between application memory and operating system data buffers, without any work done by host CPUs.

Meanwhile, 10 Gigabit Ethernet [11] defines a version of Ethernet that is ten times faster than traditional gigabit Ethernet. It offers a sockets-based interface; this is the main focus of this thesis.

Hadoop-A is an improvement of existing Hadoop project. It has solved a number of issues in Hadoop to gain better performance from the underlying system, including the serialization barrier between merge and reduce; repetitive disk access and merge. Hadoop-A has developed a C++ plug in component to overcome the aforementioned issues. A new network-levitated merge mechanism has been designed and implemented to avoid merging data and accessing disk many times, which reduces the disk bandwidth requirement and cuts down on the I/O bottleneck. A new pipeline is also introduced to cover the whole phases of data processing.

However, the new algorithm is implemented based on RDMA, which is not commonly used in commercial clusters. This restricts the good effects of Hadoop-A's novel network-levitated merge algorighm, unable to expose the benefits of reducing disk access gained from new algorithm. Thus, to make Hadoop-A readily available on any cluster,we have implemented a TCP/IP version of Hadoop-A.

Figure 1.1 can give you a clear view about the relationship of these components mentioned above. Apache Hive and Pig are two applications for dealing with large amount of data on top of Hadoop. Hadoop can support applications running on large commodity cluster and Hadoop Distributed file system provides data storage mechanism. Under Hadoop is the TCP/IP implementation of Hadoop Acceleration element which is used to improve the performance of Hadoop. It includes two components: MOFSupplier and NetMerger connected with TCP/IP socket protocol via Ethernet.

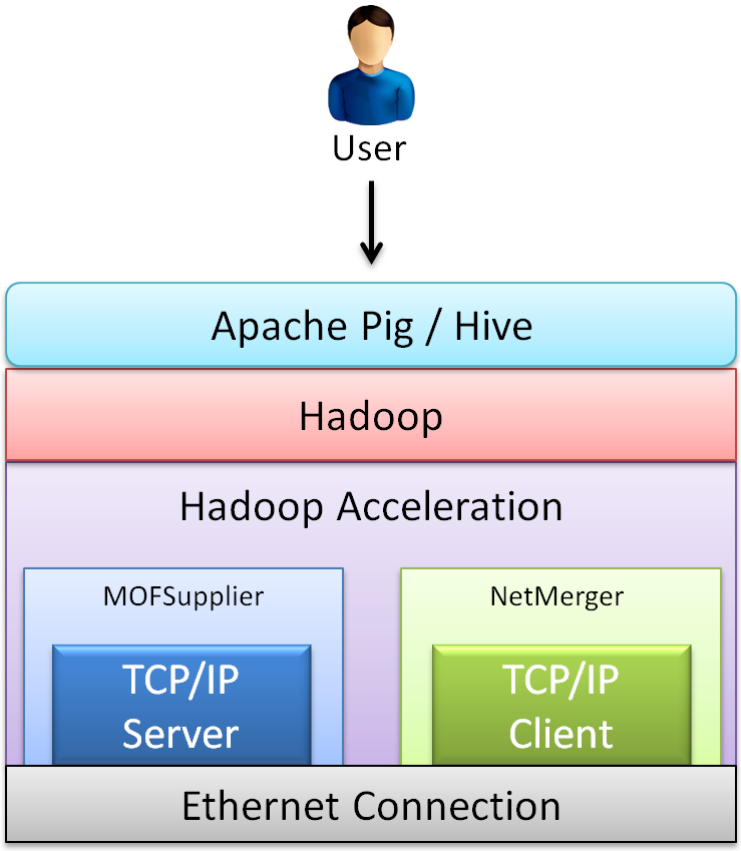

Figure 1.1: Layered framework

The rest of my thesis is orgnized as follows. Section 2 will provide a literature review and motivation. I then describe the design and implementation details of TCP/IP version of Hadoop-A in Chapter 3. Chapter 4 presents experimental results and evaluation. I will conclude my thesis in Chapter 5.

Chapter 2

Related Work

In this section, I will first introduce some details about the current status of cloud computing and the MapReduce programming model for dealing with large amount of data in cloud computing. After that, I will talk something about Hadoop, which is an open source implementation of MapReduce. By now, a lot of work has been done to improve the performance of Hadoop such as Map Reduce on-line and Hadoop-A. I will also describe them briefly and point out the exsiting problem in Hadoop-A in this chapter, which will lead to the motivation of our work. Finally, I will present Hive and Pig Latin, which are two application platform on top of Hadoop to boost the performance of distributed databases. They are good examples of successful platforms using Hadoop.

2.1 Cloud Computing

Amazon Simple Storage Service (S3) and Amazon Elastic Compute Cloud (EC2) are well-known examples. They provide Internet-based, large-scale storage space and computing resources to the users [12]. With the help of cloud computing, enterprise and personal users no longer need purchase expensive computing systems like high-performance clusters, mass storage devices, etc. All they need is a terminal (typically, operating system and web browser) with network connections to remote computing services. The terminal can either be a laptop, a desktop personal computer, or even mobile phone or other portable electronic device. Customers can pay to use the on-demand computing service in the same way as electricity, water and gas services, without knowing where they come from and how the services are managed.

2.1.1 Characteristics

Cloud computing has many characteristics [13], including: (1) Shared Infrastructure, even if the deplyment model may be different, cloud computing always uses a virtualized software model to share storage, physical services and networking capabilities; (2) Dynamic Provisioning, which automatically supplies services as required and maintains the capability to expand and contract service; (3) Network Access, we need to use PCs, laptops and some mobile devices to connect cloud servers via the Internet, where applications in the cloud range from business applications to the newest applications deployed on the smartphones; (4) Managed Metering, where metering is used to record billing information. In this way, customers are billed for services based on how much they used.

2.1.2 Service Models

Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) are three main service models in cloud computing. Software as a service means that the service provider hosts and manages the application on their own server machine, while their customer can use it over the Internet. SaaS examples include Oracle CRM On Demand and Netsuite. For platform as a service, the virtualized grid computing architecture is the basis, and it always consists of development tools, middleware and database as its infrastructure software. Through websites, developers can develop and deploy applications on this platform without considering the complexity and cost of managing and buying the underlying infrastructure. In some PaaS, it can provide some API and programming language for their users. For example, developers can write Java or Python programs on Google AppEngine. The last one is Infrastructure as a Service(IaaS), it includes the hardware (Storage, Server and network), and some low level software such as some virtual file system.

2.1.3 Deployment Models

Clouds can also be divided into private, public and hybrid (Shown in Figure 2.1). For private clouds, cloud service is used and managed within organization. In public clouds, public cloud service providers can provide management and maintenance services, and charge their customers for usage, which can simplify the implementation, users or companies can access cloud server via network. Hybrid clouds consist of many types of clouds, and allow data and/or applications to be used between one type of cloud and another, it can combine private and public cloud through their interfaces.

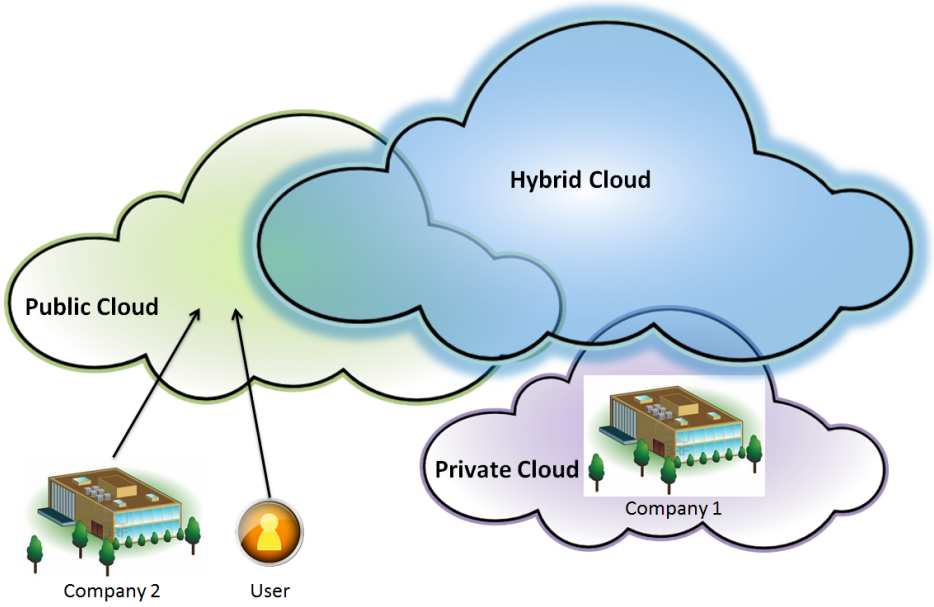

Figure 2.1: Public, Private and Hybrid Cloud Deployment

2.2 MapReduce Programming Model

The paper by Dean and Ghemawat on MapReduce in Google [1] first proposed the MapReduce Framework. Many implementations such as Hadoop are developed based on it. The purpose of their work was to facilitate the use by hiding locality optimization, load-balancing, fault-tolerance and parallelization to programming users and to show how a large number of problems can be solved with the help of MapReduce model, including sorting,

data mining, machine learning and many other issues. Last but not the least, Map Reduce is designed for large clusters consisting of tens of thousands of compute nodes. It can efficiently use compute resources suitable for dealing with large computational problems.

2.2.1 Motivation and Introduction

Nowadays, raw data is increasing dramatically. In order to deal with large amounts of various kinds of files, such as website information, and do some special-purpose computation, such as indexing, searching and getting some data analysis information about websiteis, we need a new programming model to get the input data and distribute the computation across a large number of machines, in order to finish the job in a limited time. This programming model also needs to deal with the issues of how to partition the data, parallelize the computation, and handle failures.

Map-Reduce is a new programming model to solve the above problems, which can hide the complexity of data distribution, parallel programming, fault tolerance and load balance from the user. The users only need to define a map operation, and apply it to the raw input data, which can generate some temporary key pairs. After that, they can use a reduce operation to the record which has the same key and combine the derived data appropriately.

2.2.2 Programming Model

The input of this programming model is a set of key/value pairs, and the output is also a set of key/value pairs. Users need to write Map and Reduce functions all by themselves. Map will get a pair of key/value and produce a set of temporary key/value pairs. Based on these key/value pairs, the MapReduce library can combine the record with the same key and give the result to Reduce function. In contrast, Reduce receives a key and the values, merges the values into small set of records.

There are many different implementations of the MapReduce interface. Here we will describe the MapReduce implemented on computing environments widely used at Google: where a large number of PC clusters are connected by switched Ethernet.

MapReduce can partition the input data into M pieces and many Map functions are executed parallel on different machines. Based on the partition functions, Reduce functions will partition the temporary key space into R splits. Users can specify the partition functions and the number of partitions,

Figure 2.2 presents an overview of MapReduce execution. The following actions take place when a MapReduce function is called:

1. MapReduce will first divide the data into M partitions (The size of every partition is from 16MB to 64MB) and then it will start many programs on a cluster of different machines.

2. One of them is the master program; the others are workers, which can execute their work assigned by master. Master can distribute a map task or a reduce task to an idle worker.

3. If a worker is assigned a Map task, it will parse the input data partition and output the key/value pairs, then pass the pair to a user defined Map function. The map function will buffer the temporary key/value pairs in memory

4. The pairs will periodically be written to local disk and partitioned into R pieces. After that, the local machine will inform the master of the location of these pairs.

5. If a worker is assigned a Reduce task and is informed about the location of these pairs, the Reducer will read the entire buffer by using remote procedure calls. After that, it will sort the temporary data based on the key.

6. Then, the reducer will deal with all of the records. For each key and according set of values, the reducer passes key/value pairs to a user defined Reduce function. The output is the final output of this partition.

After all of the mappers and reducers have finished their work, the master will return the result to users' programs. The output is stored in R individual files.

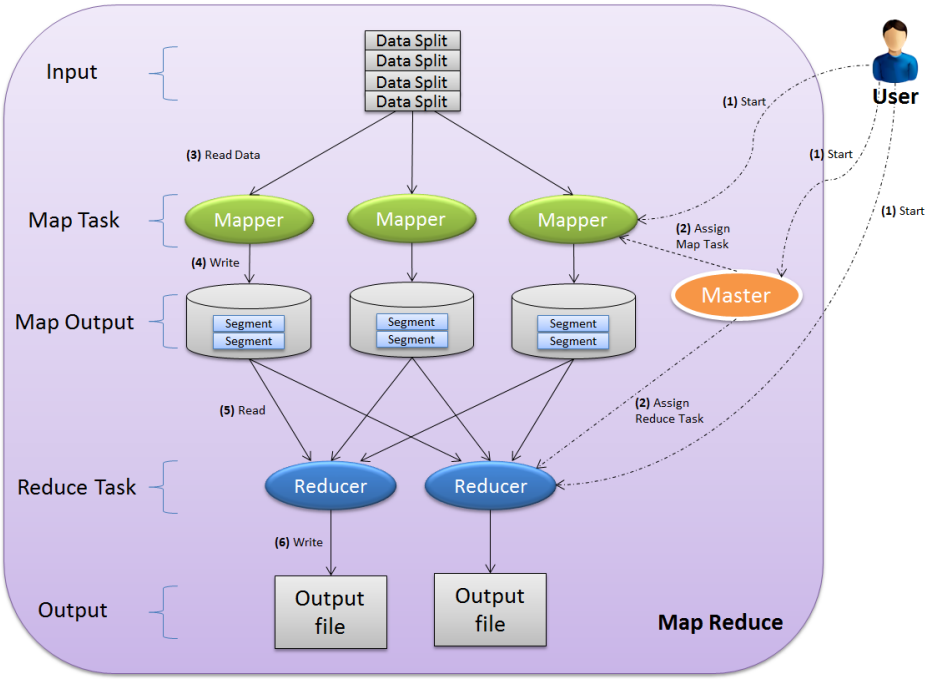

Figure 2.2: Architecture of MapReduce

2.2.3 Features of MapReduce

1) Fault Tolerance

In order to deal with large amounts of data, MapReduce needs to take advantage of hundreds or thousands of cluster nodes to do the computing. As a result, it must have the ability to handle machine failure. (1) Worker Failure: Master will periodically ping the workers. If the worker does not respond in some fixed time, the master node will mark the worker as dead. If the worker is working as a mapper, in order to deal with failure, the completed work needs to be re-executed because data is stored on the local machine. If a reducer dead, the completed work do not need to be re-executed, because their output is stored on the local machine. MapReduce can be resilient to large numbers of worker failures.

(2) Master Failure: By now, they just have one master. If master fails, they have to abort the computation.

2) Locality

MapReduce has stored data on GFS locally. GFS divides data into many blocks and each block has 3 copies. The MapReduce master gets the information about the block position and tries to allocate a map task to the node that stores the data, which can decrease the amount of network bandwidth required.

3) Backup tasks

Some machines may be very slow due to bad disks or slow CPUs, which may decrease the total performance of the MapReduce operation. In order to alleviate that problem, the master will start some backup execution of the remaining jobs when the MapReduce operation is nearly done. The task can be finished when either the primary or backup worker has finished. Experiments show that it would take 44% more time without using a backup mechanism.

2.3 Google File System and Hadoop Distributed File System

The Google File System (GFS) has the qualities to support large-scale data processing jobs on normal commercial clusters. Distributed applications can be supported by using the interface extensions provided by GFS. After the reexamination of the assumptions of traditional file system, GFS has changed the design opinion. Component failures are treated as the norm and the manners of storing huge files in GFS are optimized. By constant monitoring, replicating data and automatic recover from failure, GFS can provide the feature of fault tolerance. In order to gain high aggregate throughput from a lot of concurrent writers and readers performing all kinds of different tasks, the control of file system is separated from master. GFS can satisfy the storage requirement in Google and is widely used as the platform of storage for development and research.

Hadoop Distributed File System are developed by Yahoo! It also has a master/slave architecture which consists of a single Namenode and a number of Datanodes. The Namenode is a master server that manages the whole filesystem's namespace and controls access to files by clients. Every node in the cluster has a Datanode, which manages storage on that node. Via an RPC interface, the Namenode has the ability to open, close and rename files and directories. It also maps blocks to Datanodes. Datanodes respond for read and write requests from filesystem clients and follow Namenodes' instructions to create, delete or replicate blocks.

2.4 Hadoop

Hadoop was originally developed as infrastructure for the Nutch project [14], which crawls the web and generates a search engine index for the crawled pages. It is an open-source implementation of the MapReduce programming model, and a framework to support applications running on large commodity cluster, which transparently offer applications both data motion and reliability.

The Hadoop Map/Reduce framework has a master/slave architecture. It has only one master server/jobtracker and many slave servers/tasktrackers, one per node. Users take advantage of jobtrackers to interact with the framework. Firstly, users submit jobs to the jobtracker, which then queues the jobs and serves them on a first-come/first-served basis. The jobtracker takes charge of the allocation of map and reduce tasks to the tasktrackers. Based on the instruction of jobtracker, the tasktrackers execute tasks and deal with data motion between map and reduce phases.

2.5 MapReduce on-line

MapReduce on-line [15] have provided further improvements for Hadoop, they modified MapReduce programming model that allows data to be pipelined, which extends the MapReduce programming model beyond batch processing, decreases the execution time and

improves the utilization of system for batch jobs. In addtion, they change the version of the MapReduce framework to support on-line aggregation, which can helps users to get early returns when a job is being computed.

2.6 Hadoop-A and its TCP/IP implementation

In 2011, the paper about Hadoop-A published in SC11 have found a number of issues in Hadoop, which prevented Hadoop from achieving good performance, including a serialization barrier that delayed the reduce task, as well as repetitive disk access and merges. Hadoop-A is implemented by using a C++ plug-in component in Hadoop for data movement, which has overcome the problems mentioned above. Their novel network-levitated merge algorithm doubles throughput of data processing in Hadoop, and lowers CPU utilization by more than 36%. However, Hadoop-A is implemented based on InfiniBand, which restricts the usage of new algorithms on commercial cloud servers, and prevents them from proving their contribution towards solving the disk I/O bottleneck. On the other hand, the speed of TCP/IP-Ethernet connections is a steady technology for a long time, and is becoming faster and faster. If I can implement Hadoop-A based on the TCP/IP protocol, I can solve the problems mentioned above. In addition, I need to gain better performance on 10 Gigabit switches versus 1 Gigabit switches. The rest of the thesis illustrates how I accomplish this, by implementing TCP/IP-Ethernet support in Hadoop-A.

Chapter 3

Design and Implementation Details

Because that my work is to enable the Hadoop-A plug-in's client side and server side to connect with each other with the help of TCP/IP protocol, I need to first introduce the architecture of Hadoop-A. After that, I will present the design of my work.

3.1 Hadoop-A architecture

The Hadoop-A plug-in is implemented in C++, while the original Hadoop is implemented in Java. The main reasons for adopting C++ over Java is to avoid the Java Virtual Machine (JVM)'s overhead in processing, and to enable the RDMA connection mechanisms, which are not available in Java now. As a result, the first thing I need to consider is how to connect the C++ plug-in with Hadoop's Java modules. I will briefly describe some features of this new framework without lingering on technical details.

As we can see from figure 3.1, two new components-MOFSupplier and NetMerger are introduced to the framework of Hadoop-A. On the Java side, TaskTrackers first create a Server Socket, and then lauch a C++ side MOFSupplier and NetMerger, to build connection with them. If a connection has been created successfully, the Java side will create DataOutputStreams and DataInputStreams to store the data to/from the C++ side. On the C++ side, both MOFSuppliers and NetMergers create connections back to TaskTrackers and create streams for communication. In addition, they will add an event for down calls, which can deal with commands from Map Tasks and Reduce Tasks.

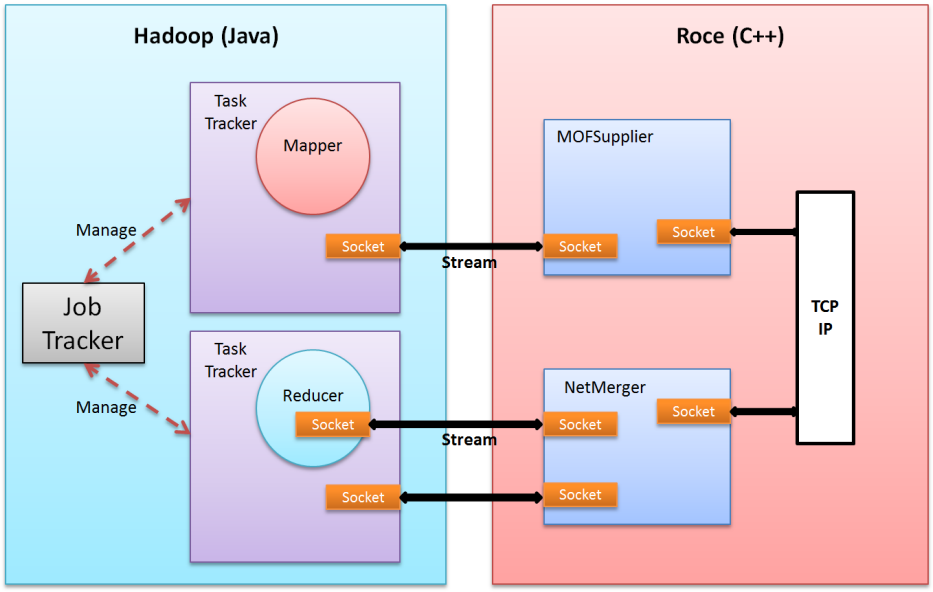

Figure 3.1: Interface between Hadoop (Java) and Roce (C++)

3.2 Interface between Hadoop (Java) and Roce (C++)

Following are details of the communication mechanism between Hadoop (Java) and Hadoop-A plug-in (Roce):

In the beginning, task trackers create socket servers to listen for connection requests. After that, NetMergers are started. Netmergers first try to connect back to the TaskTrackers. If the connection has been successfully created, netmergers will create streams for communication, and add an event for down calls, which can receive init and exit command from task tracker, Then, they create listeners to listen for new reduce tasks connection requests, once a reduce task connects to a Netmerger, the connection between Reduce Tasks and NetMergers has been established. With the help of this connection, Reduce Tasks can send requests to Netmergers, and Netmergers will return the data that it needs.

In MOFSuppliers, an event-driven thread is created to receive control commands, and inserts new Map output files (MOFs) into DataEngines. As long as MOFSuppliers receive commands from Java, it firstly check the command header; if the command is to inform that new map output has arrived, MOFSuppliers will create a new MOF entry and insert

it to the list of DataEngine which is used to cache Map Output. In order to improve the performance and take full advantage of multithreading technology, a unique thread is used to read Map Output data from disk to memory, while other threads are used to handle some other problems at the same time. Other commands can be initialization, exiting to init or exit process and threads, allocate or release resource.

Next, I will describe the details of the TCP/IP connection between MOFSuppliers and NetMergers. Many technologies are used to handle large amount of data transmission in cloud computing.

3.3 TCP/IP Implementation in Hadoop-A

In the Hadoop-A plug-in framework, since we need to deal with a big amount of data and at the same time guarantee the performance, a lot of techniques are used. Following are the methods that I employed in our TCP/IP implementation.

Epoll in Linux kernel

Epoll is an I/O event notification mechanism used in high performance network communication. It is used to replace traditional POSIX poll and select system calls. Here are some benefits of epoll over old poll/select mechanism: (1) the disadvantage of select is that the number of opened file descriptors (FD) is limited, which is sometimes not enough for the server; epoll does not have this limitation, and the largest number of FD it supports is the largest number of files that can be opened, which is much larger than 2048; (2) another disadvantage of traditional select is that when you obtain a large set of sockets, due to network delay, only some of the sockets are active, but select/poll still scans all of the socket set linearly, which can lead to efficiency proportional penalties. The problem does not exist in epoll, since it only operates on active socket, because in Linux kernel the implementation of epoll is based on fd's callback function. Only the active socket will call callback function by itself, while other idle socket will not. (3) select, poll and epoll, all require the Linux kernel

to provide information to the user space; as a result, avoiding useless memory copies is very important. Epoll solves this problem with the help of mmap via shared memory;

Multithreading

As we know, disk I/O is always the bottleneck and data movement is expensive and time consuming. Consider the case where we only use one thread to read data from disk. When we get all the data we need in the memory, we send these data to the receiver. After the receiver gets this data, it will do some calculation and write data back, every operation in sequence. Modern computers always have hardware which can support executing multiple threads efficiently, such as multi-core systems. We can make use of this multithreading technology to overlap the execution of this process. For instance, we can start a thread to read data from the disk, at the same time letting another thread send data. In the same way, we can also keep one thread receiving data while another thread computing the received data. For the purpose of increasing the speed of sending or receiving data over Ethernet, I am using multithreading to send and receive large amounts of data between Servers and Clients.

Buffer allocation management

One of the most scarce resources in computing systems is memory. Figure 3.3 shows our methods to manage buffer allocation. In MOFSuppliers, our program firstly allocation many buffers to a Memory Pool, once a Mapper write new Map Output data on the disk, disk read thread will get new empty buffer from memory pool to read data from disk. After this buffer has been fully filed, the data in this buffer will be sent to NetMerger by socket. If all of the data in this buffer has been sent out, memory pool will recycle this buffer. As long as NetMergers receive data, the receive thread get a empty buffer from Memory Pool and give this buffer with full filed data to merge thread. After all of the data in this memory block has been used, our program will recycle that memory block to get more data. In order

to avoid the overhead of allocating memory, we assign memory outside the key path of data processing, which can save a lot of execution time.

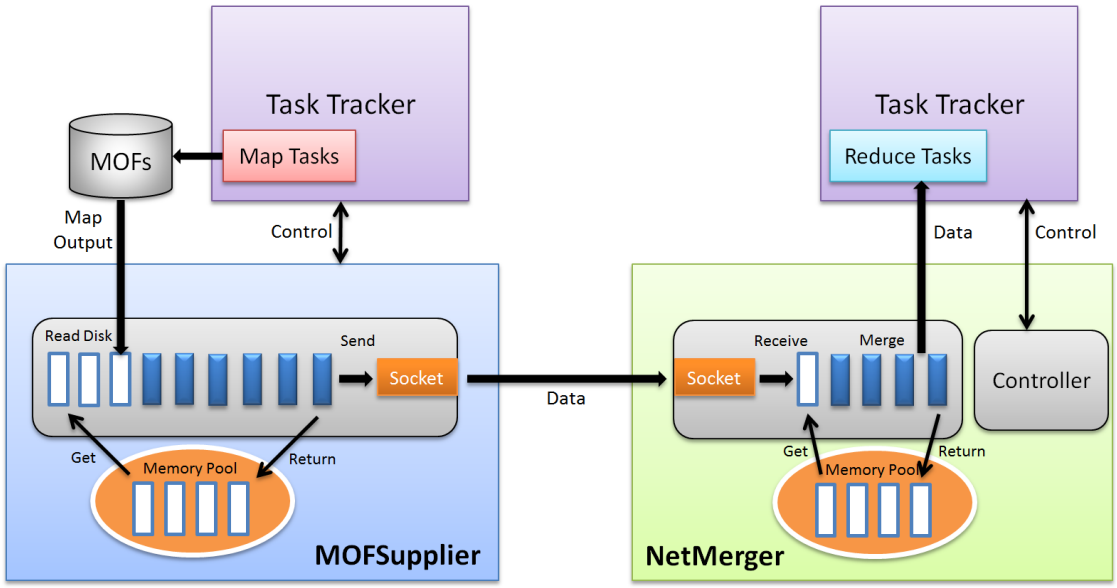

Figure 3.2: Buffer allocation management

In the following, I'd like to present the TCP/IP implementation detail to show you how I can realize the aforementioned design to achieve our goal.

3.3.1 MOFSupplier (Server)

Figure 3.3 is the main structure of MOFSupplier. We can see from the figure that there are three parts in MPFSupplier with the following functions: (1) Create a connection talking back to the Task Tracker. (2) Create a data engine, which contains a list of MOFs, and its according indexcache datacache (Implemented by DataEngine component). (3) Create an OutputServer to connect with Netmerger (Implemented by MOFServer component).

Socket connection with Task Tracker

The first function is an event-driven thread responsible for receiving control commands. The connection is created with sockets, for communicating command streams. We implement this function with the help of the epoll mechanism. After creating a connection with

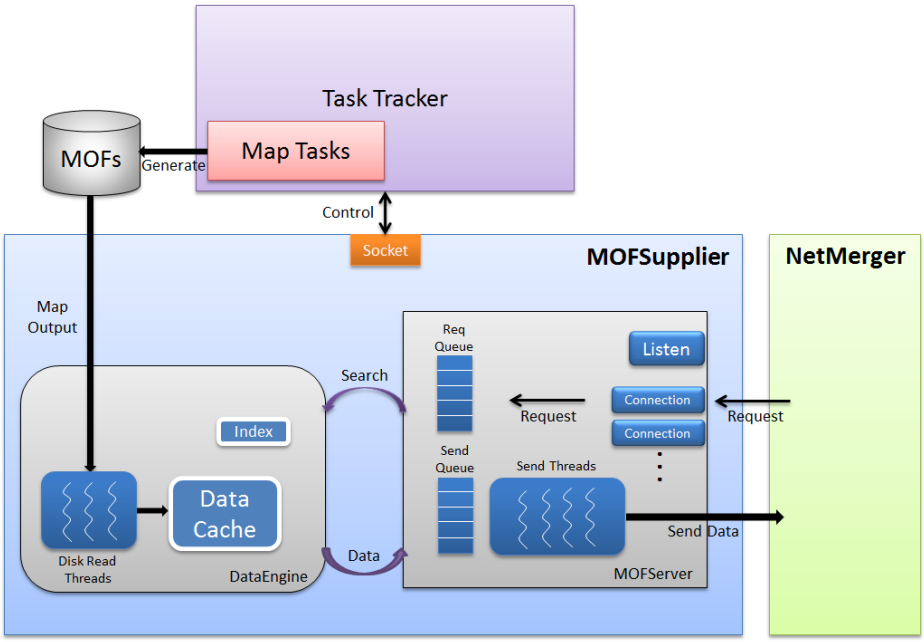

Figure 3.3: Structure of MOFSupplier

MOFSupplier, an event will be registered for down calls using epoll. When incoming messages from MOFSuppliers arrive, a thread will automatically inform the user to call down a call handler. The format of commands from Hadoop is: "NO. of (header+params) : header : param1 : param2 : ...". We separate number of (header+params), header and each parameters with colons. This format is used because we are not sure of the number and the length of parameters.

There are three types of commands. First, initiation commands are used to initialize resource and other process. Second, exit messages are called to clean jobs, quit threads and so on. The third command is "NEW_MAP_MSG", which is to notify MOFSupplier that new map output has been generated, based on the information in this command, so the DataEngine can read index and data from the disk to memory to prepare to serve NetMerger's request.

DataEngine

Next, I'd like to present the structure of DataEngine. In the beginning, DataEngine will allocate a memory pool to wait for reading data from disk into memory. As soon as MOFSupplier gets "NEW_MAP_MSG" commands, which means new Map Output is ready on the disk to be read, it will add the command information into a list and notisfy Disk Read Thread to read data into memory. In order to retrieve data quickly, DataEngine also creates an index to tell whether the data has been loaded or not, and if it has already been loaded, what's its path. As we know, disk I/O is always the bottleneck which may consume a lot of time, to gain more performance, I use one thread to read data from the disk while other threads deal with receiving and sending issues. We try to load all the data into memory, which can guarantee that DataEngine can serve others as long as it receives their request.

Another thing DataEngine needs to do is to serve the request from MOFServer. When MOFServer receives a get data request from NetMerger, it will search its index to find the required data. After that, DataEngine puts all of the information into one structure to record the position of the data and receiver information. Other threads will send data to the destination based on the message provided by the DataEngine.

MOFServer

MOFServer is the structure to receive requests from clients, search for data from the DataEngine, and finally send required data back to NetMerger. First of all, MOFServer creates a listener to accept the connection request from NetMerger. We register an event in epoll. When the connection request from NetMerger arrives, a function will be called to initialize the connection, and add connection information into a connection list. The connection list has recorded all of the connections that this MOFServer is connecting. Between each compute node pair, there is one connection. After the connection has been established, once the MOFServer received the request from NetMerger, it will add the request into a queue and inform DataEngine.

I also use data streams to send requests. The format of the request is: "jobid: mapid: mop_offset: reduceid: mem_addr: request_addr", for the same reason that the length of every parameter is unknown, we seperate them by colon. Based on the received request, DataEngine will return the data they need.

Many threads take charge of sending data from the memory to NetMergers, while at the same time another thread reads data from the disk to the memory. These sending threads are started at the initialization time of MOFServer. The number of threads is tunable. One thread serves a fixed socket fd, which can make sure all of the data from the same sender and the same receiver are in sequence. In order to achieve load balance, the number of connections in each thread is almost the same.

The size of data sent each time is larger than the default size of the socket buffer, one receive call is not always enough to receive the whole chunk which is a memory buffer allocated in the beginning. To solve this problem, MOFServer firstly send a header to inform the size of data chunk will be sent along with a lot of other informations such as: the offset of memory chunk used to receive data at the NetMerger side, this data is required by which request from NetMerger and so on. Based on the size sent by MOFSupplier, the receiver uses a while loop to call socket recv function many times, until the receiving side can get all the data specified in the header.

By now, all of the functions in MOFSupplier has been described. Next, the structure of NetMerger will be shown to you.

3.3.2 NetMerger (Client)

NetMerger is adopted to serve Reduce Tasks. Its structure is present in Figure 3.4. At the initialization time, the TaskTracker will send commands to start and init NetMerger. If some Reduce Task need to get data from MOFSupplier, it will send fetch requests to the NetMerger. NetMerger receives the requests from Reduce Tasks and send them to MOF-Supplier. As soon as required data arrives, NetMerger uses multiple threads to receive data

and pass data to another thread to do some computation (Merge). After the computation completes, threads will send data back to the Reduce Task and post more request to fetch data for computation.

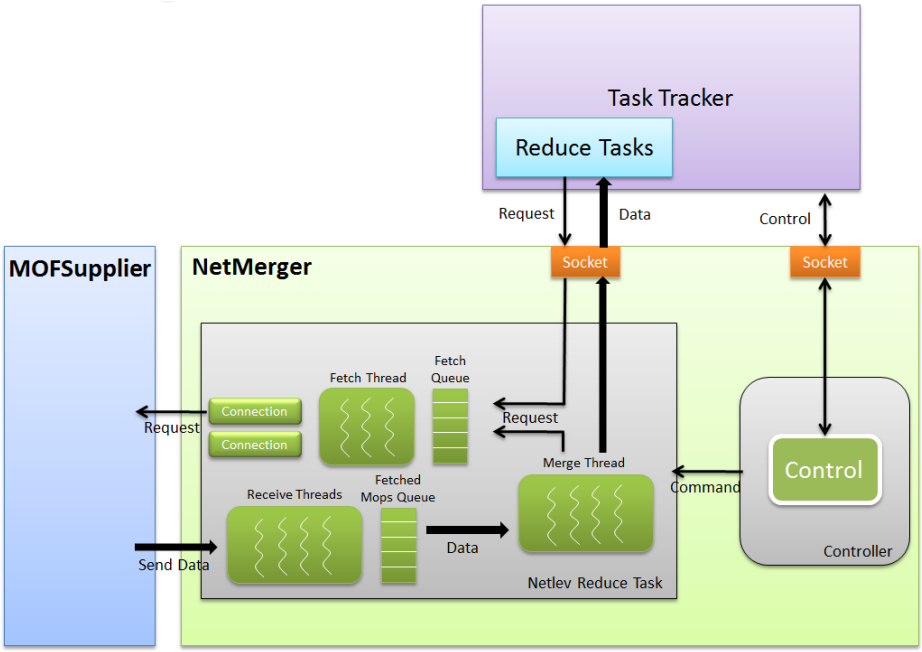

Figure 3.4: Structure of NetMerger

Socket connection with Task Tracker and Reduce Tasks

When Task Tracker begins to work, it will start NetMerger. In NetMerger, we register a client downcall handler and reduce connection handler into epoll. The client downcall handler is used to connect with Task Tracker, to receive commands from TaskTracker. These commands include inits message and exit messages. If the init messages are received, NetMerger can prepare resources for data processing. Exit messages mean you need to release memory and destroy some structure for exiting.

Controller

In Controller, it first creates an event-driven thread responsible for connecting back to the TaskTracker and receiving reducer connection requests. The function of connecting with

TaskTracker has been described above. So we'd like to tell something about dealing with Reducer connection requests. As long as the controller accepts the connection request with a Reducer, it will add the connection information into a list. Another thread is responsible for generating a new epoll set for the ReduceTask, creating a socket to receive fetch requests from the reducer and using the socket to report progress for the Reducer.

Netlev Reduce Task

If NetMerger finds that new fetch data requests have arrived from Reduce Task, it adds that request into a queue. Another thread will notice that and try to find out whether there is a connection between request sender and receiver from a connection list. If this is the first time it send a request to the receiver and there is no connection between them two, the NetMerger will send a connection request to the MOFSupplier. As long as the connection has been created, NetMerger will add the connection into the connection list and send request to MOFSupplier. The request includes which part of data it is asking for, the memory chunk that is prepared to receive data from MOFSupplier in NetMerger, the pointer to the request structure and so on.

If everything goes well, NetMerger should receive required data from MOFSupplier, after which we start many threads to receive data, and add the pointer of received data into a Fetched Mops Queue. Another thread uses received data to do some computation. We use multi-threading to overlap data receiving and data computing, which can take full advantage of modern multi-core systems to improve our implementation performance. After the merge thread finishes its job, it will send data results back to TaskTracker and add more Fetch Requests into Fetch Queue for further computation. At the end of operation, the reduce task will send an exit message to finish the job.

Above we have stated all of the components and their functions. In the following, we want to give you a clear view of data processing.

3.4 Program Flow

Figure 3.5 can give you a view of the flow of the program. I will give you a brief description based on the figure.

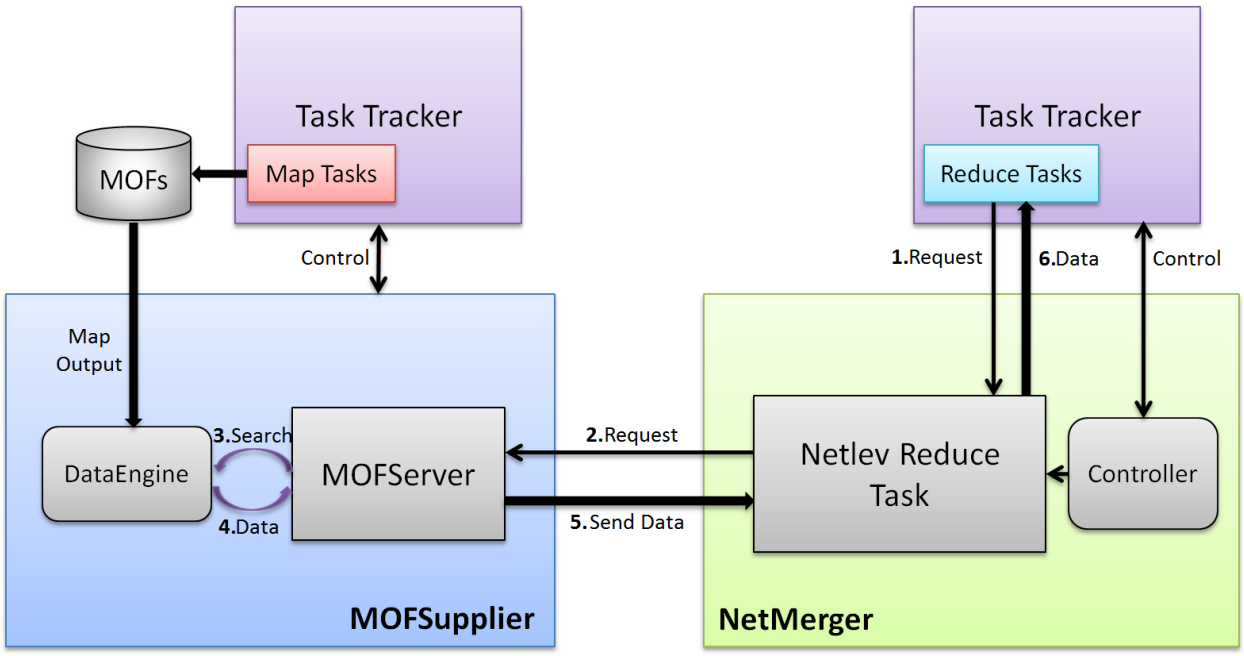

Figure 3.5: Program Flow

1. When a Reduce Task needs to fetch data from Map Task, it will send a fetch request to the NetMerger.

2. NetMerger creates a connection with MOFSupplier and sends fetchs request to MOFSupplier.

3. After receiving the request from NetMerger, MOFSupplier adds the request to the request queue, and notify DataEngine. Based on the request, DataEngine searches its Data Cache which is read from disk by the disk read thread.

4. If the required data has been found, DataEngine sends the data back to MOFServer.

5. MOFServer invokes some send threads to send data back to the NetMerger.

6. NetMerger uses many threads to receive data and gives received data to Merge Thread to do computation. As soon as computation has been finished, data will be sent to the Reduce Task.

I have introduced the detail of my TCP/IP implementation of Hadoop-A plugin. I will present the evaluation results in the following chapter.

Chapter 4

Evaluation Results

This chapter presents the evaluation of TCP/IP implementation for Hadoop-A, compared to the original Hadoop on Ethrenet.

4.1 Testbed

We do our experiments on a 26-nodes cluster. The configuration of our cluster is: Dual-socket quad-core 2.13 GHz Intel Xeon processors, 8GB of DDR2 800 MHz memory and 8x PCI-Express Gen 2.0 bus. 4 MB L2 cache is shared by four cores on one socket. We run Linux 2.6.18-164.el5 kernel on these nodes. We equipped each node with a 250GB, 7200 RPM, Western Digital SATA hard drive.

The bandwidth of RDMA is tested with the help of perf_test from OFED, while IPoIB and Gigabit Ethernet (GigE) are tested using another netperf benchmark. From 4.1, we can find that the speed of IB (RDMA) is much faster than IB (IPoIB) and Gigabit Ethernet. But it can not support Java, that's why the Hadoop-A plug-in uses C++ as its implementation language. IB (IPoIB) can achieve a bandwidth of 1078.40 MB/sec for Java and 1220.39 MB/sec for C++, while the speed of Gigabit Ethernet is only about 122.00 MB/s. As a result, if we can get better performance over Gigabit Ethernet, we will demonstrate the good effects of reducing disk access from Hadoop-A.

Table 4.1: The bandwidth of RDMA, IPoIB and Gigabit Ethernet

	RDMA	IPoIB	Gigabit Ethernet
Java	-	1078.40	122.31
C++	3239.21	1220.39	124.13

4.2 Overall Performance

First of all, we show the overall performance of my TCP/IP work compared with the original Hadoop. TeraSort and WordCount are two benchmarks for testing Hadoop programs. As mentioned in the Hadoop-A paper, we can see that Hadoop-A can get better performance on the TeraSort benchmark, but the performance of Hadoop on the WordCount benchmark is the same as for the original. So it makes no sense to run the WordCount benchmark.

Figure 4.1 shows the overall performance of running TeraSort. We are running 4G data on each node, the number of nodes is from 2 to 12. Accordingly, the largest data size we have tested is 48 GB. On each node, we run 8 Mappers and 4 Reducers. Most of the performance of TCP/IP is 20% better than original Hadoop program. For 8 nodes 32G, our TCP/IP implementation can gain 26.7% performance, which has proved the benefits of Hadoop-A's reducing disk access.

We will take 12 nodes (48GB) as an example to explain the performance of TCP/IP and the original Hadoop. As shown in Figure 4.2, the map tasks of TCP/IP are much faster, especially when the map progress is greater than 50%. The reason is that, in the map phase of TCP/IP implementation, we only execute some lightweight work.

To avoid repetitive merges, reduce tasks do not begin to merge until the completion of last Map Output (MOF) generated by map task. Once all MOFs are ready, the reduce tasks begin their work immediately. Data are fetched and merged from map tasks only for one

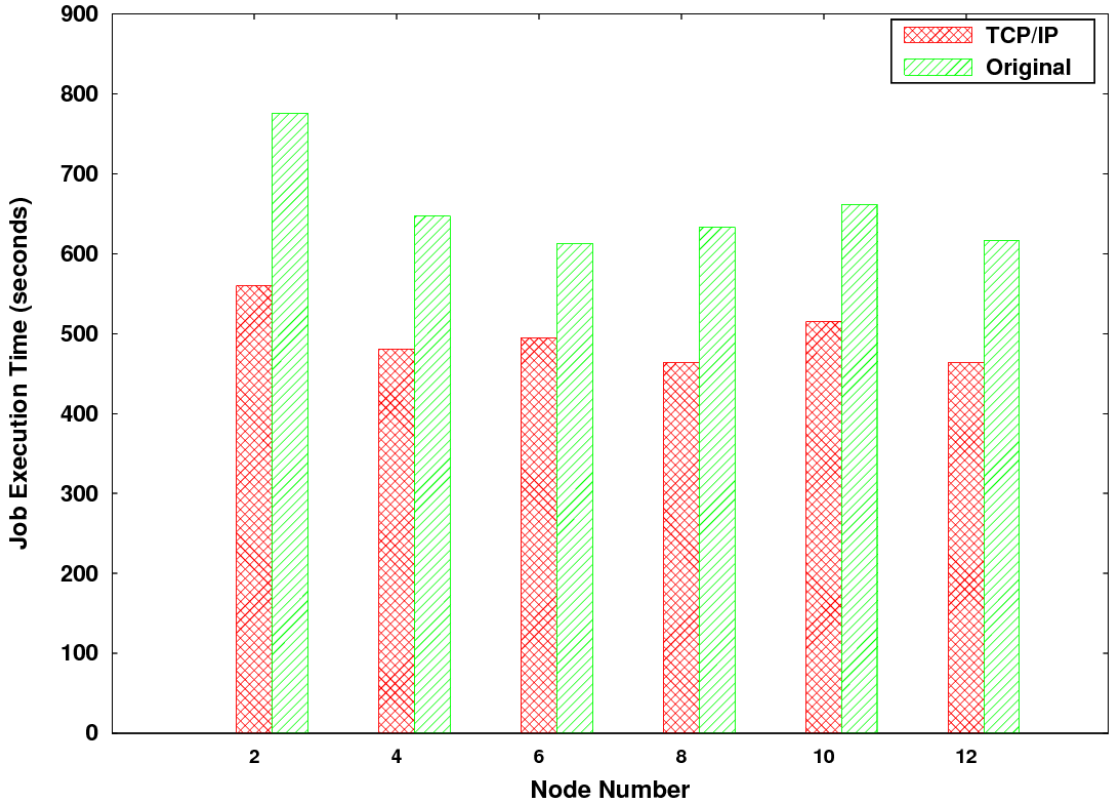

Figure 4.1: Overall performance of TCP/IP implementation

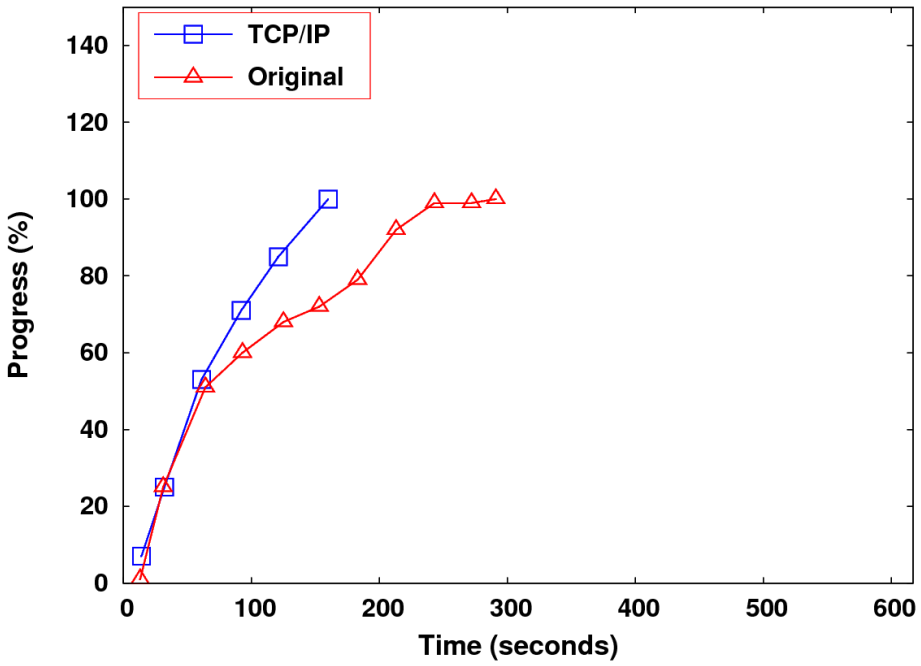

Figure 4.2: Progress of Map Task

time, which can reduce the times of disk access. From Figure 4.3 we can see reduce tasks finish their work very quickly.

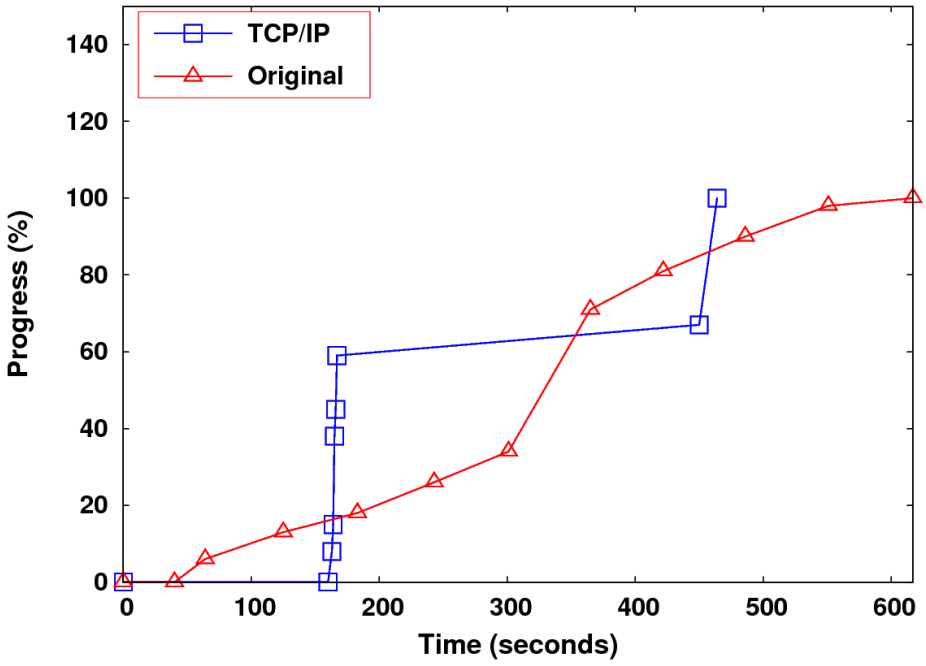

Figure 4.3: Progress of Reduce Task

4.3 Scalability

Normally in High Performance Computing (HPC), one standard of evaluating the quality of a program is scalability (scaling efficiency), which means how efficient this program can achieve when running on an increasing numbers of elements of parallel processing (CPUs / cores). Two basic ways can be used to test a given program's parallel performance, called strong scalability and weak scalability. Becasue we have more nodes to deal with data, we want to make sure the scalability of our work is good. We have examined the total execution time of Terasort's strong scalability and weak scalability.

Strong scaling is that the size of problem stays fixed while the number of CPUs/cores change, which is used to find a good situation that the problem can be solved in a reasonable time and at the same time does not waste too much resource. Usually it is hard to achieve

strong scalability due to communication overhead. To test the strong scalability of our work, we conduct our experiment on constant amount of data (24 GB) but change the number of nodes (from 2 to 12).

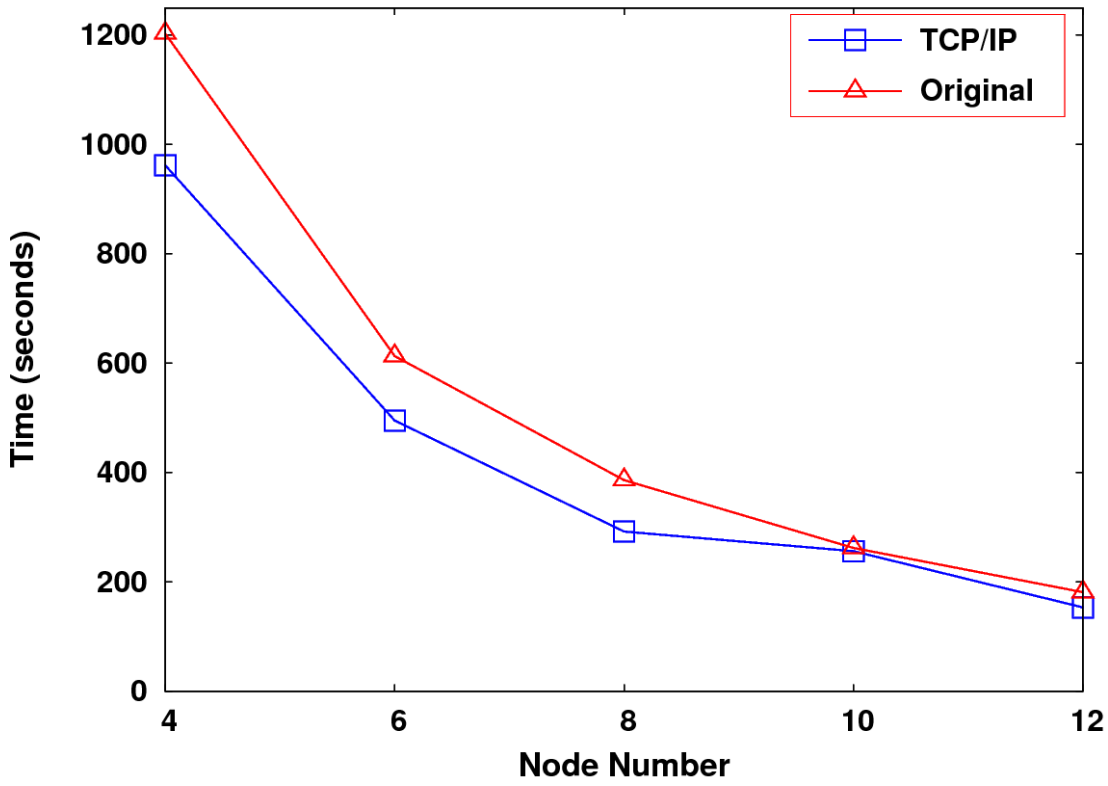

Figure 4.4: TCP/IP implementation with Increasing Number of Nodes

From Figure 4.4, we can see that, for a fixed amount of data, the TCP/IP implementation can maintain an improvement of 20% over the original Hadoop, and that both the original Hadoop and the TCP/IP implementation of Hadoop-A can achieve good scalability.

Weak scaling means for each processing element, the size of problem stays unchanged and more nodes are used to solve larger amount of problem. If the execution time stays the same, we can say the weak scalability of this program is good. To test weak scalability, each node deals with 4 GB data. The largest number of nodes used to run our program is 12, as a result, up to 48 GB data has been tested totally.

Figure 4.5 shows the weak scalability of the original Hadoop and TCP/IP implementation of Hadoop-A. Both the original Hadoop and TCP/IP implementation can achieve

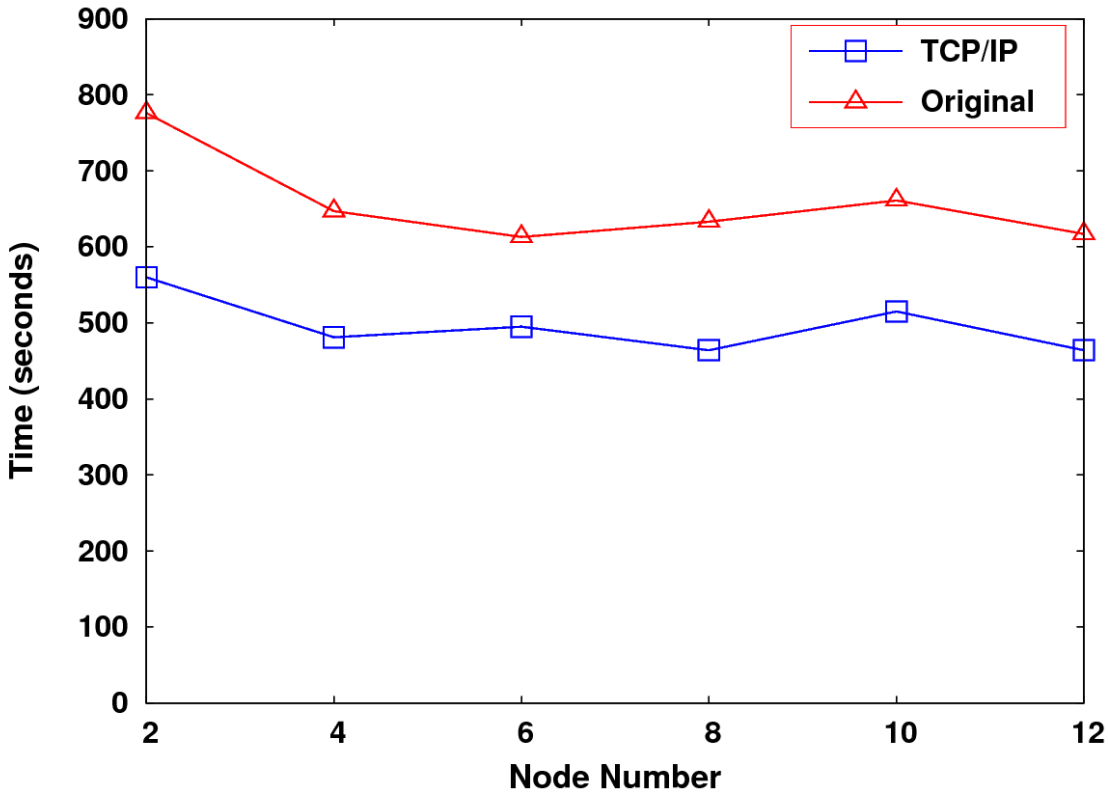

Figure 4.5: TCP/IP implementation with Increasing Data Size

good weak scalability, becasue the total excution times with these different nodes remain nearly unchanged, and as you can see, the new TCP/IP implementation can achieve better performance than original Hadoop.

Chapter 5

Conclusions

In this article, I have described the details of a TCP/IP implementation of Hadoop-A. Hadoop-A has solved a number of issues in the original Hadoop, such as the serialization barrier between merge and reduce and repetitive disk access. But that work is implemented over Infiniband (RDMA), which is not always used in commercial cloud system due to the great expense of hardware. As a result, the TCP/IP implementation of the Hadoop-A plug-in is useful for commercial cloud systems to gain good benefits from Hadoop-A.

We have presented some key technologies used to deal with large scale of data sets in TCP/IP implementation of Hadoop-A, including multi-threading, epoll mechanism and buffer allocation management. After that, we introduce two important components: MOFSupplier (Server) and NetMerger (Client). Once Netmerger gets a fetch request from Reduce task, it will send that request to the MOFSupplier. In MOFSupplier, DataEngine is a thread to read Map Output from disk, if MOFSupplier receives the fetch request from NetMerger, another thread will be woken up to send data via TCP/IP protocol. One receive thread in NetMerger receives data and the merge thread uses received data to merge. All of the buffers are allocated to a memory pool in the beginning, threads can get empty buffers from this memory pool to do their work and return them when they don't need them.

From the experimental results we find that our TCP/IP implementation can achieve 26.7% better performance than the original Hadoop project, and the scalability is good, which has fulfilled the goal of our work.

In the future, we need to test our TCP/IP implementation on much larger commercial cloud systems, which can better demonstrate the benefits of our work. In addtion, we need more test results from our implementation, including system utilization and disk read/write

bandwidth. We also want to test the performance gained when we are running applications on top of Hadoop.

Bibliography

[1] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Sixth Symp. on Operating Systen Design and Implementation, pages 137-150, December 2004.

[2] Apache Hadoop Project, http://hadoop.apache.org/.

[3] Yandong Wang, Xinyu Que, Weikuan Yu, Dror Goldenberg, Dhiraj Sehgal, "Hadoop Acceleration Through Network Levitated Merge" Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, 2011. ACM.

[4] Bisong, Anthony and Rahman, Syed (Shawon) M.., "An overview of the security concerns in enterprise cloud computing," International Journal of Network Security & Its Applications(IJNSA), Vol.3, No. 1, January

[5] Cloud computing, http://en.wikipedia.org/wiki/Cloud_computing.

[6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System," pub. 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003.

[7] K.V.Shvachko, Hairong Kuang, Sajay Radia, Robert Chansler, "The Hadoop Distributed File System," http://hadoop.apache.org/hdfs/ March, 2010.

[8] Tom White, "Hadoop The Definitive Guide," Published by OReilly Media, Inc., October 2010.

[9] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins, "Pig latin: a not-so-foreign language for data processing" In SIGMOD08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 10991110, New York, NY, USA, 2008. ACM.

[10] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang 0002, Suresh Anthony, Hao Liu, and Raghotham Murthy, "Hive - a petabyte scale data warehouse using hadoop," In ICDE, pages 9961005, 2010.

[11] Apache Nutch, http://en.wikipedia.org/wiki/10_Gigabit_Ethernet

[12] Danish Jamil, Hassan Zaki, "Cloud Computing Security," Danish Jamil et al./ International Journal of Engineering Science and Technology (IJEST)

[13] Dialogic Corporation, “Introduction to Cloud Computing,” www.dialogic.com/.

[14] Apache Nutch, http://nutch.apache.org/.

[15] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears, “MapReduce Online” In NSDI, pages 312328, April 2010.