Implementation Issues of A Cloud Computing Platform

Bo Peng, Bin Cui and Xiaoming Li
Department of Computer Science and Technology, Peking University
{pb,bin.cui,lxm}@pku.edu.cn

Abstract

Cloud computing is Internet based system development in which large scalable computing resources are provided "as a service" over the Internet to users. The concept of cloud computing incorporates web infrastructure, software as a service (SaaS), Web 2.0 and other emerging technologies, and has attracted more and more attention from industry and research community. In this paper, we describe our experience and lessons learnt in construction of a cloud computing platform. Specifically, we design a GFS compatible file system with variable chunk size to facilitate massive data processing, and introduce some implementation enhancement on MapReduce to improve the system throughput. We also discuss some practical issues for system implementation. In association of the China web archive (Web InfoMall) which we have been accumulating since 2001 (now it contains over three billion Chinese web pages), this paper presents our attempt to implement a platform for a domain specific cloud computing service, with large scale web text mining as targeted application. And hopefully researchers besides our selves will benefit from the cloud when it is ready.

1 Introduction

As more facets of work and personal life move online and the Internet becomes a platform for virtual human society, a new paradigm of large-scale distributed computing has emerged. Web-based companies, such as Google and Amazon, have built web infrastructure to deal with the internet-scale data storage and computation. If we consider such infrastructure as a "virtual computer", it demonstrates a possibility of new computing model, i.e., centralize the data and computation on the "super computer" with unprecedented storage and computing capability, which can be viewed as a simplest form of cloud computing.

More generally, the concept of cloud computing can incorporate various computer technologies, including web infrastructure, Web 2.0 and many other emerging technologies. People may have different perspectives from different views. For example, from the view of end-user, the cloud computing service moves the application software and operation system from desktops to the cloud side, which makes users be able to plug-in anytime from anywhere and utilize large scale storage and computing resources. On the other hand, the cloud computing service provider may focus on how to distribute and schedule the computer resources. Nevertheless, the storage and computing on massive data are the key technologies for a cloud computing infrastructure.

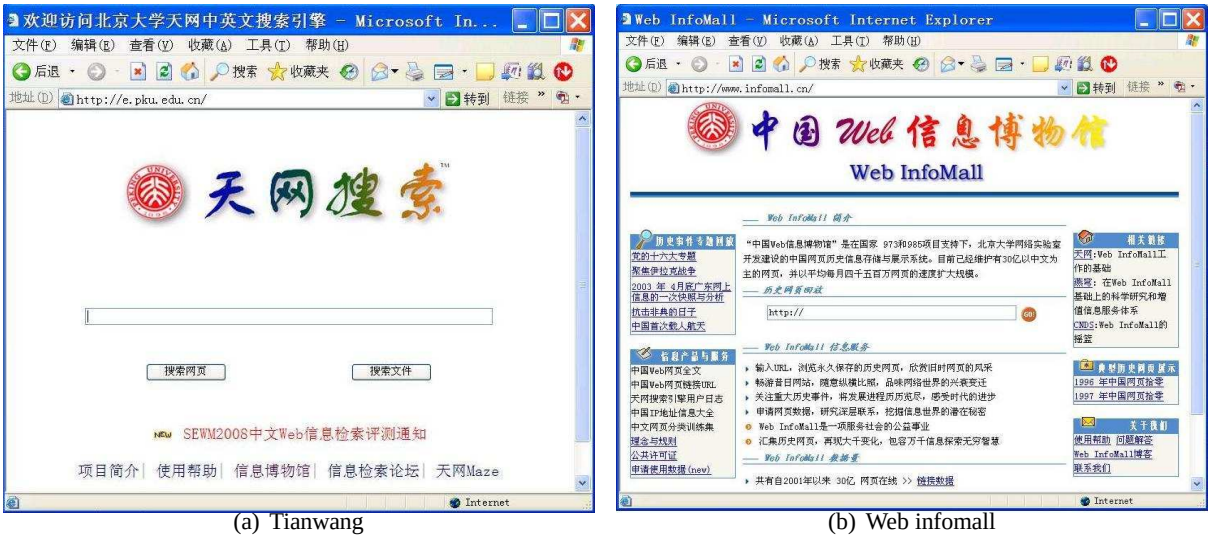
Google has developed its infrastructure technologies for cloud computing in recent years, including Google File System (GFS) [8], MapReduce [7] and Bigtable [6]. GFS is a scalable distributed file system, which

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

emphasizes fault tolerance since it is designed to run on economically scalable but inevitably unreliable (due to its sheer scale) commodity hardware, and delivers high performance service to a large number of clients. Bigtable is a distributed storage system based on GFS for structured data management. It provides a huge three-dimensional mapping abstraction to applications, and has been successfully deployed in many Google products. MapReduce is a programming model with associated implementation for massive data processing. MapReduce provides an abstraction by defining a "mapper" and a "reducer". The "mapper" is applied to every input key/value pair to generate an arbitrary number of intermediate key/value pairs. The "reducer" is applied to all values associated with the same intermediate key to generate output key/value pairs. MapReduce is an easy-to-use programming model, and has sufficient expression capability to support many real world algorithms and tasks. The MapReduce system can partition the input data, schedule the execution of program across a set of machines, handle machine failures, and manage the inter-machine communication.

More recently, many similar systems have been developed. KosmosFS [3] is an open source GFS-Like system, which supports strict POSIX interface. Hadoop [2] is an active Java open source project. With the support from Yahoo, Hadoop has achieved great progress in these two years. It has been deployed in a large system with 4,000 nodes and used in many large scale data processing tasks.

In Oct 2007, Google and IBM launched "cloud computing initiative" programs for universities to promote the related teaching and research work on increasingly popular large-scale computing. Later in July 2008, HP, Intel and Yahoo launched a similar initiative to promote and develop cloud computing research and education. Such cloud computing projects can not only improve the parallel computing education, but also promote the research work such as Internet-scale data management, processing and scientific computation. Inspired by this trend and motivated by a need to upgrade our existing work, we have implemented a practical web infrastructure as cloud computing platform, which can be used to store large scale web data and provide high performance processing capability. In the last decade, our research and system development focus is on Web search and Web Mining, and we have developed and maintained two public web systems, i.e., *Tianwang* Search Engine [4] and Web Archive system *Web infomall* [1] as shown in Figure 1.



(a) Tianwang (b) Web infomall

Figure 1: Search engine and Chines web archive developed at SEWM group of PKU

During this period, we have accumulated more than 50 TB web data, built a PC cluster consisting of 100+ PCs, and designed various web application softwares such as webpage text analysis and processing. With the increase of data size and computation workload in the system, we found the cloud computing technology is a promising approach to improve the scalability and productivity of the system for web services. Since 2007, we

started to design and develop our web infrastructure system, named “Tplatform”, including GFS-like file system “TFS” [10] and MapReduce computing environment. We believe our practice of cloud computing platform implementation could be a good reference for researchers or engineers who are interested in this area.

2 TPlatform: A Cloud Computing Platform

In this section, we briefly introduce the implementation and components of our cloud computing platform, named “Tplatform”. We first present the overview of the system, followed by the detailed system implementation and some practical issues.

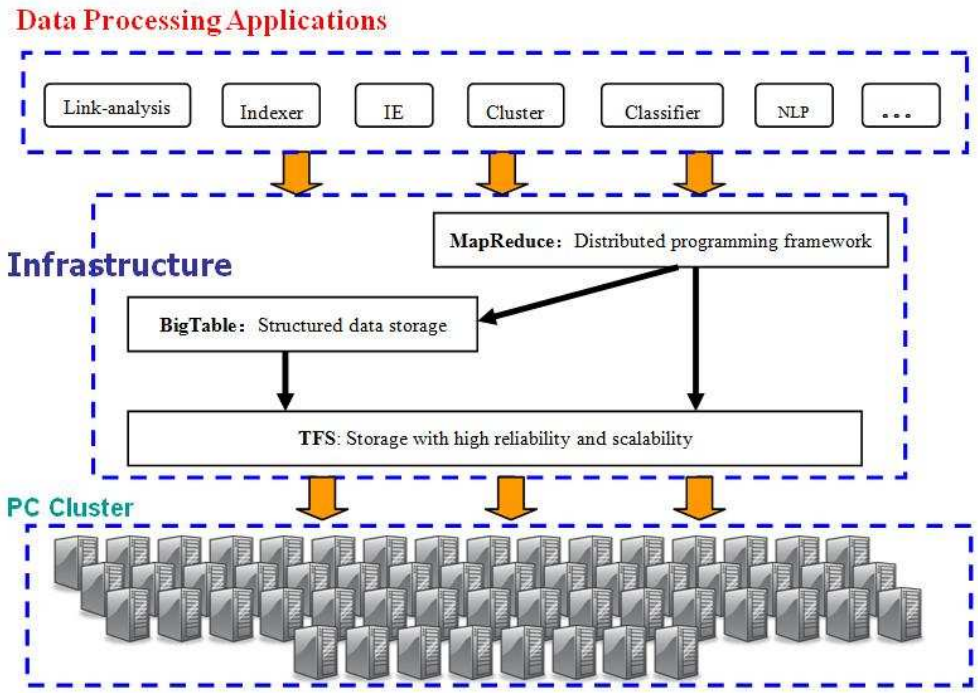


Figure 2: The System Framework of Tplatform

Fig 2 shows the overall system framework of the “Tplatform”, which consists of three layers, i.e., PC cluster, infrastructure for cloud computing platform, and data processing application layer. The PC cluster layer provides the hardware and storage devices for large scale data processing. The application layer provides the services to users, where users can develop their own applications, such as Web data analysis, language processing, cluster and classification, etc. The second layer is the main focus of our work, consisting of file system TFS, distributed data storage mechanism BigTable, and MapReduce programming model. The implementation of BigTable is similar to the approach presented in [6], and hence we omit detailed discussion here.

2.1 Implementation of File System

The file system is the key component of the system to support massive data storage and management. The designed TFS is a scalable, distributed file system, and each TFS cluster consists of a single master and multiple chunk servers and can be accessed by multiple client.

2.1.1 TFS Architecture

In TFS, files are divided into variable-size chunks. Each chunk is identified by an immutable and globally unique 64 bit chunk handle assigned by the master at the time of chunk creation. Chunk servers store the chunks on the local disks and read/write chunk data specified by a chunk handle and byte range. For the data reliability, each chunk is replicated on multiple chunk servers. By default, we maintain three replicas in the system, though users can designate different replication levels for different files.

The master maintains the metadata of file system, which includes the namespace, access control information, the mapping from files to chunks, and the current locations of chunks. It also controls system-wide activities such as garbage collection of orphaned chunks, and chunk migration between chunk servers. Each chunk server periodically communicates with the master in HeartBeat messages to report its state and retrieve the instructions.

TFS client module is associated with each application by integrating the file system API, which can communicate with the master and chunkservers to read or write data on behalf of the application. Clients interact with the master for metadata operations, but all data-bearing communication goes directly to the chunkservers.

The system is designed to minimize the master's involvement in file accessing operations. We do not provide the POSIX API. Besides providing the ordinary read and write operations, like GFS, we have also provided an atomic record appending operation so that multiple clients can append concurrently to a file without extra synchronization among them. In the system implementation, we observe that the record appending operation is the key operation for system performance. We design our own system interaction mechanism which is different from GFS and yields better record appending performance.

2.1.2 Variable Chunk Size

In GFS, a file is divided into fixed-size chunks (e.g., 64 MB). When a client uses record appending operation to append data, the system checks whether appending the record to the last chunk of a certain file may make the chunk overflowed, i.e., exceed the maximum size. If so, it pads all the replica of the chunk to the maximum size, and informs the client that the operation should be continued on the new chunk. (Record appending is restricted to be at most one-fourth of the chunk size to keep worst case fragmentation at an acceptable level.) In case of write failure, this approach may lead to duplicated records and incomplete records.

In our TFS design, the chunks of a file are allowed to have variable sizes. With the proposed system interaction mechanism, this strategy makes the record appending operation more efficient. Padding data, record fragments and record duplications are not necessary in our system. Although this approach brings some extra cost, e.g., every data structure of chunk needs a chunk size attribute, the overall performance is significantly improved, as the read and record appending operations are the dominating operations in our system and can benefit from this design choice.

2.1.3 File Operations

We have designed different file operations for TFS, such as read, record append and write. Since we allow variable chunk size in TFS, the operation strategy is different from that of GFS. Here we present the detailed implementation of read operation to show the difference of our approach.

To read a file, the client exchanges messages with the master, gets the locations of chunks it wants to read from, and then communicates with the chunk servers to retrieve the data. Since GFS uses the fixed chunk size, the client just needs to translate the file name and byte offset into a chunk index within the file, and sends the master a request containing the file name and chunk index. The master replies with the corresponding chunk handle and locations of the replicas. The client then sends a request to one of the replicas, most likely the closest one. The request specifies the chunk handle and a byte range within that chunk. Further reads of the same chunk do not require any more client-master interaction unless the cached information expires or the file is reopened.

In our TFS system, the story is different due to the variable chunk size strategy. The client can not translate the byte offset into a chunk index directly. It has to know all the sizes of chunks in the file before deciding which chunk should be read. Our solution is quite straightforward, when a client opens a file using read mode, it gets all the chunks' information from the master, including chunk handle, chunk size and locations, and use these information to get the proper chunk. Although this strategy is determined by the fact of variable chunk size, its advantage is that the client only needs to communicate with the master once to read the whole file, which is much efficient than GFS' original design. The disadvantage is that when a client has opened a file for reading, later appended data by other clients is invisible to this client. But we believe this problem is negligible, as the majority of the files in web applications are typically created and appended once, and read by data processing applications many times without modifications. If in any situation this problem becomes critical, it can be easily overcome by set an expired timestamp for the chunks' information and refresh it when invalid.

The TFS demonstrates our effort to build an infrastructure for large scale data processing. Although our system has the similar assumptions and architectures as GFS, the key difference is that the chunk size is variable, which makes our system able to adopt different system interactions for record appending operation. Our record appending operation is based on chunk level, thus the aggregate record appending performance is no longer restricted by the network bandwidth of the chunk servers that store the last chunk of the file. Our experimental evaluation shows that our approach significantly improves the concurrent record appending performance for single file by 25%. More results on TFS have been reported in [10]. We believe the design can apply to other similar data processing infrastructures.

2.2 Implementation of MapReduce

MapReduce system is another major component of the cloud computing platform, and has attracted more and more attentions recently [9, 7, 11]. The architecture of our implementation is similar to Hadoop [2], which is a typical master-worker structure. There are three roles in the system: Master, Worker and User. Master is the central controller of the system, which is in charge of data partitioning, task scheduling, load balancing and fault tolerance processing. Worker runs the concrete tasks of data processing and computation. There exist many workers in the system, which fetch the tasks from Master, execute the tasks and communicate with each other for data transfer. User is the client of the system, implements the Map and Reduce functions for computation task, and controls the flow of computation.

2.2.1 Implementation Enhancement

We make three enhancements to improve the MapReduce performance in our system. First, we treat intermediate data transfer as an independent task. Every computation task includes map and reduce subtasks. In a typical implementation such as Hadoop, reduce task starts the intermediate data transfer, which fetches the data from all the machines conducting map tasks. This is an uncontrollable all-to-all communication, which may incur network congestion, and hence degrade the system performance. In our design, we split the transfer task from the reduce task, and propose a "Data transfer module" to execute and schedule the data transfer task independently. With appropriate scheduling algorithm, this method can reduce the probability of network congestion. Although this approach may aggravate the workload of Master when the number of transfer tasks is large, this problem can be alleviated by adjusting the granularity of transfer task and integrating data transfer tasks with the same source and target addresses. In practice, our new approach can significantly improve the data transfer performance.

Second, task scheduling is another concern on MapReduce system, which helps to commit resources between a variety of tasks and schedule the order of task execution. To optimize the system resource utility, we adopt multi-level feedback queue scheduling algorithm in our design. Multiple queues are used to allocate the concurrent tasks, and each of them is assigned with a certain priority, which may vary for different tasks with respect to the resources requested. Our algorithm can dynamically adjust the priority of running task, which

balances the system workload and improves the overall throughput.

The third improvement is on data serialization. In MapReduce framework, a computation task consists of four steps: map, partition, group and reduce. The data is read in by map operation, intermediate data is generated and transferred in the system, and finally the results are exported by reduce operation. There exist frequent data exchanges between memory and disk which are generally accomplished by data serialization. In our implementation of MapReduce system, we observed that the simple native data type is frequently used in many data processing applications. Since memory buffer is widely used, most of the data already reside in the memory before they are de-serialized into a new data object. In other words, we should avoid expensive de-serialization operations which consume large volume of memory space and degrade the system performance. To alleviate this problem, we define the data type for key and value as void* pointer. If we want to de-serialize the data with native data type, a simple pointer assignment operation can replace the de-serialization operation, which is much more efficient. With this optimization, we can also sort the data directly in the memory without data de-serialization. This mechanism can significantly improve the MapReduce performance, although it introduces some cost overhead for buffer management.

2.2.2 Performance Evaluation on MapReduce

Due to the lack of benchmark which can represent the typical applications, performance evaluation on MapReduce system is not a trivial task. We first use PennySort as the simple benchmark. The result shows that the performance of intermediate data transfer in the shuffle phase is the bottle neck of the system, which actually motivated us to optimize the data transfer module in MapReduce. Furthermore, we also explore a real application for text mining, which gathers statistics of Chinese word frequency in webpages. We run the program on a 200GB Chinese Web collection. Map function analyzes the content of web page, and produces every individual Chinese word as the key value. Reduce function sums up all aggregated values and exports the frequencies. In our testbed with 18 nodes, the job was split into 3385 map tasks, 30 reduce tasks and 101550 data transfer tasks, the whole job was successfully completed in about 10 hours, which is very efficient.

2.3 Practical Issues for System Implementation

The data storage and computation capability are the major factors of the cloud computing platform, which determine how well the infrastructure can provide services to end users. We met some engineering and technical problems during the system implementation. Here we discuss some practical issues in our work.

2.3.1 System Design Criteria

In the system design, our purpose is to develop a system which is scalable, robust, high-performance and easy to be maintained. However, some system design issues may be conflicted, which places us in a dilemma in many cases. Generally, we take three major criteria into consideration for system design: 1) For a certain solution, what is bottleneck of the procedure which may degenerate the system performance? 2) Which solution has better scalability and flexibility for future change? 3) Since network bandwidth is the scarce resource of the system, how to fully utilize the network resource in the implementation? In the following, we present an example to show our considerations in the implementation.

In the MapReduce system, fault tolerance can be conducted by either master or workers. Master takes the role of global controller, maintains the information of the whole system and can easily decide whether a failed task should be rerun, and when/where to be rerun. Workers only keep local information, and take charge of reporting the status of running tasks to Master. Our design combines the advantages of these two factors. The workers can rerun a failed task for a certain number of times, and are even allowed to skip some bad data records which cause the failure. This distributed strategy is more robust and scalable than centralized mechanism, i.e., only re-schedule failed tasks in the Master side.

2.3.2 Implementation of Inter-machine Communication

Since the implementation of cloud computing platform is based on the PC cluster, how to design the inter-machine communication protocol is the key issue of programming in the distributed environment. The Remote Procedure Call (RPC) middle ware is a popular paradigm for implementing the client-server model of distributed computing, which is an inter-process communication technology that allows a computer program to cause a subroutine or procedure to execute on another computer in a PC cluster without the programmer explicitly coding the details for this remote interaction. In our system, all the services and heart-beat protocols are RPC calls. We exploit Internet Communications Engine (ICE), which is an object-oriented middleware that provides object-oriented RPC, to implement the RPC framework. Our approach performs very well under our system scale and can support asynchronous communication model. The network communication performance of our system with ICE is comparable to that of special asynchronous protocols with socket programming, which is much more complicated for implementation.

2.3.3 System Debug and Diagnosis

Debug and Diagnosis in distributed environment is a big challenge for researchers and engineers. The overall system consists of various processes distributed in network, and these processes communicate each other to execute a complex task. Because of the concurrent communications in such system, many faults are generally not easy to be located, and hence can hardly be debugged. Therefore, we record complete system log in our system. In All the server and client sides, important software boundaries such as API and RPC interfaces are all logged. For example, log for RPC messages can be used to check integrality of protocol, log for data transfer can be used to validate the correctness of transfer. In addition, we record performance log for performance tuning. In our MapReduce system, log in client side records the details of data read-in time, write-out time of all tasks, time cost of sorting operation in reduce task, which are tuning factors of our system design.

In our work, the recorded log not only helps us diagnose the problems in the programs, but also helps find the performance bottleneck of the system, and hence we can improve system implementation accordingly. However, distributed debug and diagnosis are still low efficient and labor consuming. We expect better tools and approaches to improve the effectiveness and efficiency of debug and diagnosis in large scale distributed system implementation.

3 Conclusion

Based on our experience with Tplatform, we have discussed several practical issues in the implementation of a cloud computing platform following Google model. It is observed that while GFS/MapReduce/BigTable provides a great conceptual framework for the software core of a cloud and Hadoop stands for the most popular open source implementation, there are still many interesting implementation issues worth to explore. Three are identified in this paper.

- The chunksize of a file in GFS can be variable instead of fixed. With careful implementation, this design decision delivers better performance for read and append operations.
- The data transfer among participatory nodes in reduce stage can be made ”schedulable” instead of ”uncontrolled”. The new mechanism provides opportunity for avoiding network congestions that degrade performance.
- Data with native types can also be effectively serialized for data access in map and reduce functions, which presumably improves performance in some cases.

While Tplatform as a whole is still in progress, namely the implementation of BigTable is on going, the finished parts (TFS and MapReduce) are already useful. Several applications have shown the feasibility and advantages of our new implementation approaches. The source code of Tplatform is available from [5].

Acknowledgment

This work was Supported by 973 Project No. 2007CB310902, IBM 2008 SUR Grant for PKU, and National Natural Science foundation of China under Grant No.60603045 and 60873063.

References

[1] *China Web InfoMall*. http://www.infomall.cn, 2008.

[2] *The Hadoop Project*. http://hadoop.apache.org/, 2008.

[3] *The KosmosFS Project*. http://kosmosfs.sourceforge.net/, 2008.

[4] *Tianwang Search*. http://e.pku.edu.cn, 2008.

[5] *Source Code of Tplatform Implementation*. http://net.pku.edu.cn/˜ webg/tplatform, 2009.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 15–15, 2006.

[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI '04: Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, pages 137–150, 2004.

[8] G. Sanjay, G. Howard, and L. Shun-Tak. The google file system. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 29–43, 2003.

[9] H. Yang, A. Dasdan, R. Hsiao, and D. S. Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040, 2007.

[10] Z. Yang, Q. Tu, K. Fan, L. Zhu, R. Chen, and B. Peng. Performance gain with variable chunk size in gfs-like file systems. In *Journal of Computational Information Systems*, pages 1077–1084, 2008.

[11] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI '07: Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, pages 29–42, 2007.