The Efficient Data Storage Management System Using BW Transform on Cluster-based Private Cloud Data Center

Cho Cho Khaing, Thinn Thu Naing
University of Computer Studies, Yangon
chokhaing.ck@gmail.com,ucsy21@most.gov.mm

Abstract

The widespread popularity of Cloud computing as a preferred platform for the deployment of web applications has resulted in an enormous number of applications moving to the cloud, and the huge success of cloud service providers. The data center storage management plays a vital role in cloud computing environments. Especially the PC cluster-based data storage is necessary to manage data on low cost storage servers in which storage space can be reduced. This system presents an efficient data storage approach to push work out to many nodes in a cluster using Hadoop File System (HDFS) with variable chunk size to facilitate massive data processing. This system introduces the implementation enhancement on MapReduce to improve the system throughput and the scalability to keep on working with the amount of existing physical storage capacity when the number of users and files increase.

1. Introduction

Information Technology (IT) organizations worldwide are dealing with the tremendous growth of data. With the growth of capacity comes the complexity of managing the storage for that data. In addition to the continued growth in capacity, the accelerated use of virtual servers and desktops is rapidly altering the storage landscape. IT organizations worldwide are turning to virtualized environments to improve datacenter flexibility and scalability. This in turn drives implementation of networked storage solutions, which can create new pressures on storage performance as I/Os that were previously more distributed are aggregated into a smaller number of host interconnects. There are also implications for organizations' data protection processes and architectures to ensure that every virtual server is protected and that the storage has the same flexibility and resiliency as the virtualized server environment.

When a dataset outgrows the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines. Filesystems that manage the storage across a network of machines are called distributed filesystems. Since they are network-based, all the complications of network programming kick in, thus making distributed filesystems more complex than regular disk filesystems. For example, one of the biggest challenges is making the filesystem tolerate node failure without suffering data loss. Hadoop comes with a distributed filesystem called HDFS, which stands for Hadoop Distributed Filesystem. Hadoop is a very powerful platform for processing and coordinating the movement of data across various architectural components. Its only drawback is that the primary computing model is MapReduce.

This paper presents the efficient storage system on private cloud. This approach has been designed to use over virtualized storage system. The idea of the proposed system is to exploit the

use of virtualization technology to avoid unnecessary storage purchases and reduce storage space for addressing large volumes of data handling problem.

The rest of this paper is organized as follows. Section 2 describes the related work. In section 3, cloud computing is discussed. Section 4 the proposed system is presented. Finally section 5 concludes the paper.

2. Related Work

There has been some recent work on bringing together ideas from MapReduce and HDFS system; however, this work focuses mainly on language and interface issues. A.Verma, N.Zea, B.Cho, I.Gupta, and R.H.Campbell [2] showed that their approach can achieve better performance times than a traditional MapReduce framework. Their experiments with Hadoop demonstrated speedups of up to 87% for well-suited applications, and an average of 25% for more typical applications. HadoopDB [1] built a hybrid system that takes the best features from the parallel DBMS and Hadoop approaches the prototype they built approaches parallel databases in performance and efficiency, scalability, fault tolerance, and flexibility of MapReduce-based systems. This paper [5] presented a multi-GPU parallel volume rendering implemention built using the MapReduce programming model and gave implementation details of the library, including specific optimizations made for our rendering and compositing design. They showed that our system scales with respect to the size of the volume, and (given enough work) the number of GPUs. This paper [8] gave an overview of MapReduce programming model and its applications. It described the workflow of MapReduce process. Some important issues, like fault tolerance, are studied in more detail. It also took a look at the different implementations of MapReduce. In this paper [4], they presented EUCALYPTUS, an opensource software framework for cloud computing that implements what is commonly referred to as Infrastructure as a Service (IaaS); systems that give users the ability to run and control entire virtual machine instances deployed across a variety physical resources. The scientific workflow framework[3] that supported streams as first-class data, and is optimized for performant and reliable execution across desktop and Cloud platforms. This paper presented the workflow framework features and its empirical evaluation on the Eucalyptus cloud. In this paper, they showed the need for streaming support in scientific workflows to support the next generation of scientific and engineering applications that respond to events in the environment at real time. They proposed a data model for streams that can coexist with collections and files that are currently supported by workflows.

3. Cloud Computing

Cloud computing is Internet-based computing, whereby shared resources, software and information are provided to computers and other devices on-demand, like the electricity grid. Cloud computing describes a new supplement, consumption and delivery model for IT services based on the Internet, and it typically involves the provision of dynamically scalable and often virtualized resources as a service over the Internet. Typical cloud computing providers deliver common business applications online which are accessed from another web service or software like a web browser, while the software and data are stored on servers. Most cloud computing infrastructure consists of reliable services delivered through data centers and built on servers. Clouds often appear as single points of access for all consumers' computing needs.

4. Proposed System

In my proposed system, there are four main functions: chunking block, splitting block using HDFS, MapReduce process and compression. MapReduce process includes map function, combine function, partition function, compute hash function and reduce function.

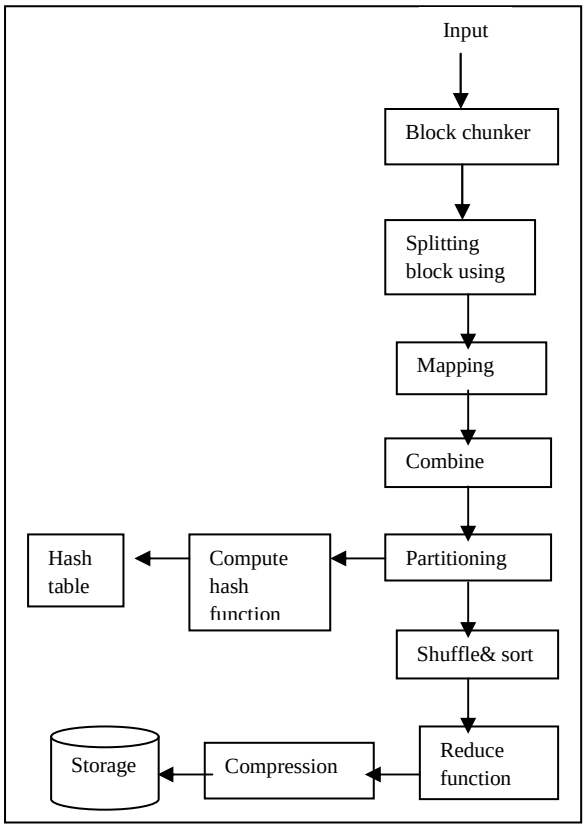

Figure 1. System Architecture of the proposed system

4.1 Chunking Blocks

In this chunking blocks, we will divide fixed sized blocks for text files and variable sized blocks for databases.

4.2. The Design of HDFS

HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters on commodity hardware.

There are Hadoop clusters running today that store petabytes of data. Streaming data access HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. A dataset is typically generated or copied from source, then various analyses are performed on that dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

Hadoop doesn't require expensive, highly reliable hardware to run on. It's designed to run on clusters of commodity hardware for which the chance of node failure across the cluster is high, at least for large clusters. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure. Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS. HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency. HBase is currently a better choice for low-latency access. Since the namenode holds filesystem metadata in memory, the limit to the number of files in a filesystem is governed by the amount of memory on the namenode. As a rule of thumb, each file, directory, and block takes about 150 bytes. Files in HDFS may be written to by a single writer. Writes are always made at the end of the file. There is no support for multiple writers, or for modifications at arbitrary offsets in the file.

4.3. MapReduce process

MapReduce is a programming model for data processing. The model is simple, yet not too simple to express useful programs in. Hadoop can run MapReduce programs written in various languages: Java, Ruby, Python, and C++. Most importantly, MapReduce programs are inherently parallel, thus putting very large-scale data analysis into the hands of anyone with enough machines at their disposal.

One of the most significant advantages of MapReduce is that it provides an abstraction that hides many system-level details from the programmer. Therefore, a developer can focus on what computations need to be performed, as opposed to how those computations are actually carried out or how to get the data to the processes that depend on them. Like OpenMP and MPI, MapReduce provides a means to distribute computation without burdening the programmer with the details of distributed computing (but at a different level of granularity). However, organizing and coordinating large amounts of computation is only part of the challenge. Large-data processing by definition requires bringing data and code together for computation to occur no small feat for datasets that are terabytes and perhaps petabytes in size. MapReduce addresses this challenge by providing a simple abstraction for the developer, transparently handling most of the details behind the scenes in a scalable, robust, and efficient manner. Instead of moving large amounts of data around, it is far more efficient, if possible, to move the code to the data. This is operationally realized by spreading data across the local disks of machines in a cluster and running processes on machines that hold the data. The complex task of managing storage in such a processing environment is handled by the distributed file system that underlies MapReduce.

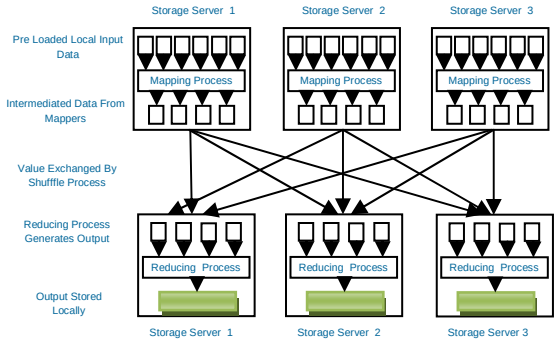

Figure 2. MapReduce process flow of proposed system

4.3.1. Partition

Partitioners are responsible for dividing up the intermediate key space and assigning intermediate key-value pairs to reducers. In other words, the partitioner specifies the node to which an intermediate key-value pair must be copied. The partitioner involves computing the hash value of the key and then taking the mod of that value with the number of reducers. This assigns approximately the same number of keys to each reducer (dependent on the quality of the hash function). The partitioner determines which reducer will be responsible for processing a particular key, and the execution framework uses this information to copy the data to the right location during the shuffle and sort phase.

4.3.2. Combine

Combiners are an optimization in MapReduce that allow for local aggregation before the shuffle and sort phase. It emits a key-value pair for each word in the collection. Furthermore, all these key-value pairs need to be copied across the network. Combiner performs local aggregation on the output of each mapper to compute a local count for a word over all the documents processed by the mapper. With this modification, the number of intermediate key-

value pairs will be at most the number of unique words in the collection times the number of mappers. Each combiner operates in isolation and therefore does not have access to intermediate output from other mappers. Like the reducer, the combiner is provided keys and all values associated with each key. It can emit any number of key-value pairs, but the keys and values must be of the same type as the reducer.

4.3.3. Computing Hashing method

Choosing a good hash function is of the utmost importance. An **uniform** hash function is one that equally distributes data items over the whole hash table data structure. If the hash function is poorly chosen data items may tend to **clump** in one area of the hash table and many collisions will ensue. A non-uniform dispersal pattern and a high collision rate cause an overall data structure performance degradation. There are several strategies such as division method, multiplication method and etc. for maximizing the uniformity of the hash function and thereby maximizing the efficiency of the hash table. These above methods are only effective when all data item keys are of the same, fixed size (in bits). To hash non-fixed length data item keys, we uses **variable string addition** method which is used to hash variable length strings.

Variable String Addition Method

BEGIN
PROCEDURE HASH
Accept input string
Calculate the sum of input string according to ASCII code
Divide this result by 256
END PROCEDURE

Figure 3. Variable String Addition hashing algorithm

To hash a variable-length string, The hash function works by first summing the ASCII value of each character in the variable length strings, modulo 256, to a total. A hash value, range 0-255, is computed.
Eg. String s="I am a student"
h(s)=73+32+97+109+32+97+32+115+116+117+
100+101+110+116=1247%256=223.

Therefore, h(s)=223, so we save this block into 223 index in hash table.

Hash Table

In our proposed system, the hash table uses separate chaining to resolve collisions. It is implemented as an array of linked lists. To insert an item into the hash table, it is appended to one of the linked lists. The linked list to which it is appended is determined by hashing that item. The constructor takes a single argument of type unsigned int which specifies the size of hash table desired. The constructor simply initializes the HashTable base class and the array member variable accordingly. Initializing the array variable involves constructing the required number of emptylinked lists. The running time for the ChainedHashTable constructor is *O(M)* where *M* is the size of the hash table. The ChainedHashTable destructor calls the Purge member function.

4.4. Compression

The Burrows-Wheeler transform is applied on blocks of input data (symbols). It is usually the case that larger blocks result in greater compressibility of the transformed data at the expense of time and system resources. One of the effects of BWT is to produce blocks of data with more and longer runs (strings of identical symbols) than those found in the original data. The increasing the number of runs and their lengths tends to improve a the compressibility of data. By additionally applying Move-to-Front coding, the data will be in a format which is generally more compressible by even zero order

statistical encoders such as traditional implementations of Huffman coding.

The first step of BWT is to read in a block of N symbols $C_0 \ldots C_{N-1}$.

Example:
Read in the following block: " I am a student."

N = 15
C_0 = 'i'
C_1 = ' '
…
C_{13} = 't'
C_{14} = '.'

The next step is to think of the block as a cyclic buffer. N strings (rotations) $S_0 \ldots S_{N-1}$ may be constructed such that:
$S_0 = C_0, \ldots, C_{N-1}$
$S_1 = C_1, \ldots, C_{N-1}, C_0$
$S_2 = C_2, \ldots, C_{N-1}, C_0, C_1$
…
$S_{N-1} = C_{N-1}, C_0, \ldots, C_{N-2}$

$S_{N-1} = C_{N-1}, C_0, \ldots, C_{N-2}$

Example:
"i am a student." yields the following rotations:

S_0 = "i am a student."
S_1 = " am a student.i"
S_2 = "am a student.i "
S_3 = "m a student.i a"
S_4 = " a student.i am"
S_5 = "a student.i am "
S_6 = " student.i am a"
S_7 = "student.i am a "
S_8 = "tudent.i am a s"
S_9 = "udent.i am a st"
S_{10} = "dent.i am a stu"
S_{11} = "ent.i am a stud"
S_{12} = "nt.i am a stude"
S_{13} = "t.i am a studen"
S_{14} = ".i am a student"

The third step of BWT is to lexicographically sort $S_0 \ldots S_{N-1}$.

Example:
"i am a student." yields the following sorted rotations:

S_4 = " a student.i am"
S_1 = " am a student.i"
S_6 = " student.i am a"
S_{14} = ".i am a student"
S_5 = "a student.i am "
S_2 = "am a student.i "
S_{10} = "dent.i am a stu"
S_{11} = "ent.i am a stud"
S_0 = "i am a student."
S_3 = "m a student.i a"
S_{12} = "nt.i am a stude"
S_7 = "student.i am a "
S_{13} = "t.i am a studen"
S_8 = "tudent.i am a s"
S_9 = "udent.i am a st"

The final step in the transform is to output a string, L, consisting of the last character in each of the rotations in their sorted order along with I, the sorted row containing S_0.

Example:
"I am a student." yields the following output:

L = "miat ud.ae nst", I = 14

Move-To-Front Coding

Move-To-Front (MTF) coding is a technique that encodes streams of symbols based on an adapting code. Imagine that symbols are encoded by their position in a list of every symbol in the alphabet. Initially the list is in a lexicographical (or some predetermined) order. Once a symbol in a stream is encoded, that symbol is moved from it's current position to the front of the list and its former predecessors are moved one position back.

Example:
Given an alphabet of ‘ ’, ‘.’, ‘a’, ‘d’, ‘e’, ‘i’, ‘m’, ‘n’, ‘s’, ‘t’, ‘u’ encode "miat ud.ae nst":

Renaming Symbols	**Symbol List**	**Encoded Symbols**
miat ud.ae nst	‘ ’, ‘.’, ‘a’, ‘d’, ‘e’, ‘i’, ‘m’,‘n’,‘s’, ‘t’, ‘u’	
iat ud.ae nst	‘m’, ‘ ’, ‘.’, ‘a’, ‘d’, ‘e’, ‘i’, ‘n’, ‘s’, ‘t’, ‘u’	6
at ud.ae nst	‘i’, ‘m’, ‘ ’, ‘.’, ‘a’, ‘d’, ‘e’, ‘n’, ‘s’, ‘t’, ‘u’	6,6
t ud.ae nst	‘a’, ‘i’, ‘m’, ‘ ’, ‘.’, ‘d’, ‘e’, ‘n’, ‘s’, ‘t’, ‘u’	6,6,4
ud.ae nst	‘t’, ‘a’, ‘i’, ‘m’, ‘ ’,‘.’, ‘d’, ‘e’, ‘n’, ‘s’, ‘u’	6,6,4,9
ud.ae nst	‘ ’, ‘t’, ‘a’, ‘i’, ‘m’, ‘.’, ‘d’, ‘e’, ‘n’, ‘s’, ‘u’	6,6,4,9,4
d.ae nst	‘u’, ‘ ’, ‘t’, ‘a’, ‘i’, ‘m’, ‘.’, ‘d’, ‘e’, ‘n’, ‘s’,	6,6,4,9,4,10
.ae nst	‘d’, ‘u’, ‘ ’, ‘t’, ‘a’, ‘i’, ‘m’, ‘.’, ‘e’, ‘n’, ‘s’	6,6,4,9,4,10,7
ae nst	‘.’, ‘d’, ‘u’, ‘ ’, ‘t’, ‘a’, ‘i’, ‘m’, ‘e’, ‘n’, ‘s’	6,6,4,9,4,10,7,7
e nst	‘a’, ‘.’, ‘d’, ‘u’, ‘ ’, ‘t’, ‘i’, ‘m’, ‘e’, ‘n’, ‘s’	6,6,4,9,4,10,7,7,5
nst	‘e’, ‘a’, ‘.’, ‘d’, ‘u’, ‘ ’, ‘t’, ‘i’, ‘m’, ‘n’, ‘s’	6,6,4,9,4,10,7,7,5,8
nst	‘ ’, ‘e’, ‘a’, ‘.’, ‘d’, ‘u’, ‘t’, ‘i’, ‘m’, ‘n’, ‘s’	6,6,4,9,4,10,7,7,5,8,5
st	‘n’, ‘ ’, ‘e’, ‘a’, ‘.’, ‘d’, ‘u’, ‘t’, ‘i’, ‘m’, ‘s’	6,6,4,9,4,10,7,7,5,8,5,9
t	‘s’, ‘n’, ‘ ’, ‘e’, ‘a’, ‘.’, ‘d’, ‘u’, ‘t’,	6,6,4,9,4,10,7,7,5,8,5,9,10

	‘i’, ‘m’	
	‘t’, ‘s’, ‘n’, ‘ ’, ‘e’, ‘a’, ‘.’, ‘d’, ‘u’, ‘i’, ‘m’	6,6,4,9,4,10,7,7,5,8,5,9,10 ,8

There are more than one time for each number.
6-2,4-2,9-2,10-2,7-2,5-2,8-2. There has same frequency.

Huffman Coding

After MTF coding, this output is encoded by Huffman coding as the following:
4-2-00
5-2-111
6-2-110
7-2-101
8-2-100
9-2-011
10-2-010
Therefore, the final binary result of compression is:
110 110 00 011 00 010 101 101 111 100 111 011 010 100.

5. Conclusion

When the amount of data are growing in existing storage capacity, the proposed system will improve the storage utilization and efficiency. By using MapReduce model with BW Transform, we will reduce the amount of data redundancy and will expect to save storage space and increase speed. The proposed system will reduce the total amount of data required to be moved over the network and get significant performance benefits as well as high throughput. This proposed system will provide the scalability to keep on working with the amount of existing physical storage capacity when the number of users and files increase. Moreover, this system will enable the service provider to add a new data centers as needed, while existing data centers continue functioning without interruption.

References

[1]A.Abouzeid1,K.B.Pawlikowski,D.Abadi,A.Silberschatz, A.Rasin:“ *HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads*”, VLDB ‘09, August 24-28, 2009, Lyon, France Copyright 2009 VLDB Endowment, ACM .

[2] A.Verma, N.Zea, B.Cho, I.Gupta, and R.H.Campbell:“ *Breaking the MapReduce Stage Barrier”*, University of Illinois at Urbana-Champaign,2009.

[3]D.Zinn,Q.Hart,T.McPhillips,B.L.ascher,Y.Simmhan, M.Giakkoupis, V.K.Prasanna:“ *Towards Reliable, Performant Workflows for Streaming-Applications on Cloud Platforms*”, University of California,2010.

[4] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov:“ *The Eucalyptus Open-source Cloud-computing System*”, Computer Science Department, University of California, Santa Barbara, 2008.

[5] J.A. Stuart, C.K. Chen, K.L. Ma:“ *Multi-GPU Volume Rendering using MapReduce*”, MAPREDUCE 2010 Chicago, Illinois USA Copyright 2010 ACM.

[6] J D.E. ATKINS ET AL., “*Revolutionizing Science and Engineering Through Cyberinfrastructure: Report of the National Science FoundationBlue-ribbonAdvisory Panel on Cyberinfrastructure*”, NSF, Report of theNational Science FoundationBlue-ribbonAdvisory Panel on Cyberinfrastructure, January 2003.

[7] S. Scully and W.Benjamin: “*Improving Storage Efficiencies with Data Deduplication and Compression*” May 2010.

[8] T. Aarnio:“ *Parallel data processing with MapReduce*”, Helsinki University of Technology, TKK T-110.5190 Seminar on Internetworking,2009.