ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

Improving the performance of Google file system to support Big Data

Ms. Pooja Mittal[1], Munesh Kataria[2]

Assistant Professor, Department of Computer Science & Application, M.D. University, Rohtak[1]

M.Tech Student, Department of Computer Science & Application, M.D. University, Rohtak[2]

Abstract: After doing research on Google File System, we find out some methods to improve the performance of Google file system. Google File System is a scalable distributed file system for large size distributed data-intensive applications. It provides high fault tolerance while running on inexpensive commodity hardware and it delivers high aggregate performance to a large number of clients. But there are some limitations in it such as it uses same chunk size to append and write data. Fixed chunk size decreases its performance for append data. So we will explain some methods to increase its performance by changing some attributes of typical Google File System. This paper is divided into five parts. First part presents the basic introduction of Google File System, second part provides the performance of GFS cluster for a 64 MB chunk size, third part shows the performance of real time GFS clusters, fourth part presents a method to increase the performance of GFS, and finally part fifth concludes the effect of variable size chunk on GFS.

Keywords: Availability, chunk, performance, reliability, scalability.

I. INTRODUCTION

Big Data is a distributed file system which is developed by Google Inc. for their own use. It was designed to provide reliable and efficient access to data using large clusters of commodity hardware. It was initially implemented to satisfy Google's core data storage and usage needs and mainly for search engines. It was based on "Big Files", developed by Larry Page and Sergey Brin. In it, files were divided in fixed size 64 megabyte chunks, same as sectors and clusters in regular file system. Files are extremely rarely overwritten; they are mainly appended to or read. GFS mainly uses cheap, "commodity" computers, so failure rate is generally high but throughput is also high.

GFS mainly have two types of nodes: one master node and large number of chunkserver nodes. Every file is divided into fixed size chunks which are stored on chunkservers and assigned a unique 64 bit label by master at their creation time. Every chunk is replicated several times according to their end-in demand but it must be replicated at least 3 times. Master node doesn't store data chunks, it has metadata about chunks such as their 64 –bit label, their copies location and what processes are reading and writing them. Master node also replicate a chunk when number of copies become less than three. All this data on Master node is periodically updated by "a Heart-beat Message" from chunks. Master node grant permission to any process for a limited time interval to modify any chunk and then modified chunkserver, which is always primary chunk holder, do the changed to other chunkservers having backup copies. Changes are saved when acknowledgement came from every chunkserver with backup copy. Any process first of all query the master node for desired chunks location, if the chunk are not in use then Master node provide the location and program then request and receive the data from the chunkserver directly.GFS is provided as a userspace library , it is not implemented in kernel of an operating system.

II. PERFORMANCE OF GFS

A typical Google File System cluster may contain hundreds of chunkservers and clients. But when we measured performance on a GFS cluster consisting of one master with two master replicas, 16 chunkservers, and 16 clients then the results were following:

A. Reads

When N clients read simultaneously from given file system then each client reads a randomly selected 4 MB region from a large 320 GB file set. It was repeated 256 times so that each client ends up reading 1 GB of data. When the chunkservers taken together have only 32 GB of memory, so we expected at most a 10% hit rate in the Linux buffer cache. The results should be close to cold cache results. Figure1 shows aggregate read rate for N clients and its given theoretical limit. This limit peaks at an aggregate of 125 MB/s when 1 Gbps link between the two switches were saturated, or 12.5 MB/s per client when its 100 Mbps network interface got saturated, whichever applied. The observed read rate is 10 MB/s, or 80% of the per-client limit, when just one client was reading. The aggregate read rate reached 94 MB/s, nearly 75% of the 125 MB/s link limit, for all 16 readers (6 MB/s per client). The efficiency dropped from 80% to 75% because as the number of readers increases, the probability that multiple readers simultaneously read from the same chunkserver also increases.

DOI 10.17148/IJARCCE.2015.4522

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

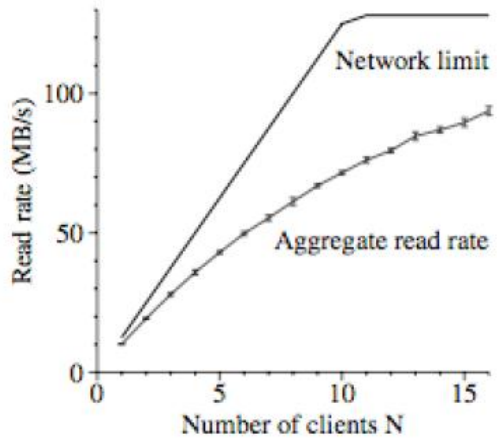

Fig 1 Read Performance

B. Writes

When M clients write simultaneously to *M* distinct files then each client write 1 GB of data to a new file as a series of 1 MB writes. The average write rate and its theoretical limit are shown in Figure2. The overall limit plateaus at 67 MB/s because we need to write each byte to at least 3 of 16 chunk servers, each with 12.5 MB/s input connection. The overall write rate for one client was 6.3 MB/s, about half of the limit. The main culprit for this was network stack. It did not interact very well with the pipelining scheme which we use for pushing data to chunk replicas. Total delays in propagating the data from one replica to another replica reduced the overall write rate. Aggregate write rate reached 35 MB/s for 16 clients, about half the given theoretical limit. As in the case of reads, it becomes more likely that multiple clients write concurrently to the same chunkserver as the number of clients increases. Moreover, collisions are more likely for 16 writers than for 16 readers because each write involves three different replicas. Writes were slower than what we would like. In real time this has not been a major problem because even though it increases the latencies as seen by individual clients, it did not significantly affect the aggregate write bandwidth delivered by the system to a large number of clients.

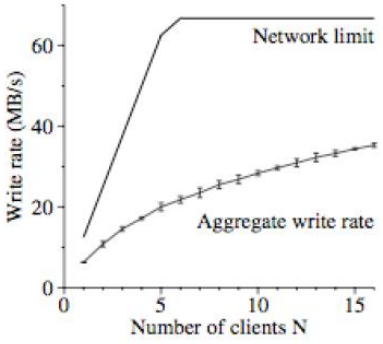

Fig 2 Write Performance

C. *Record Appends*

Figure shows the record append performance for GFS when *M* clients append simultaneously to a single file. Here we shows the record append performance for GFS when *M* clients append simultaneously to a single file. The performance was limited by the network bandwidth of the chunkservers that store last chunk of the file to append, independent of the number of the clients. It started at 6.0 MB/s for one client and dropped to 4.8 MB/s for 16 clients, mostly due to the congestion and variances in network transfer rates seen by the different clients.

These applications tend to produce multiple files concurrently. Or we can say, *M* clients append to *N* shared files simultaneously where both *N* and *M* are in dozens or may be in hundreds. Thus, the chunkserver network congestion in experiment was not a significant issue in practice because a client can make progress on writing one file while the chunkservers for another file are busy.

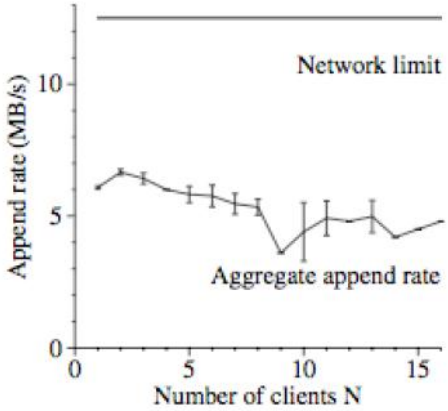

Fig 3 Record Append Performance

III. PERFORMANCE FOR REAL TIME CLUSTER

According to a research by Sanjay Ghemawat and his team, the performances for two real time cluster are as follows:

Cluster	A	B
Chunkservers	342	227
Available Disk Cap.	72 TB	180 TB
Used Disk Cap	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead Files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at Chunkservers	13 GB	21 GB
Metadata at Master	48 MB	60 MB

Here we have two fair sized storage systems in which one is utilizing nearly 80% of available space and another is utilizing nearly 90% of available space.

Here we find that chunk metadata appears to scale linearly with number of chunks. A has average file size about 1/3 of B. A has average file size nearly 75 MB and B has 210 MB which is much larger than average data center file size. Here we get some performance data for the two clusters:

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

Cluster	A	B
Read Rate – last minute	583 MB/s	380 MB/s
Read Rate – last hour	562 MB/s	384 MB/s
Read Rate – since restart	589 MB/s	49 MB/s
Write Rate – last minute	1 MB/s	101 MB/s
Write Rate – last hour	2 MB/s	117 MB/s
Write Rate – since restart	25 MB/s	13 MB/s
Master Ops – last minute	325 Op/s	533 Op/s
Master Ops – last hour	381 Op/s	518 Op/s
Master Ops – since restart	202 Op/s	347 Op/s

GFS has excellent sequential read performance, also very good sequential write performance, but unimpressive small write performance. When we see performance of cluster A's, in last minute it performs about 125 small writes, averaging about 8k each. So not good for oracle call centre with 500 desk. But it is not bad for a system constructed from commodity hardware.

IV. METHOD TO IMPROVE PERFORMANCE

A. Variable chunk size

When a GFS cluster with one master, nine chunkservers, and ten clients are given and the master server and chunkserver machines are Dell 2850 configured with two 2.800GHz Intel Xeon processors, 2.0GB of memory, six 7200 rpm Ultra SCSI drives configured as one software RAID-0 volume. All given client machines are same as above except the memory is 4GB. And all of these machines have 1 Gbps full-duplex Ethernet connection to a Dell 2748 1 Gbps switch.

a) ***Master Operation***

When ten clients concurrently execute the sequence with 42000 operations then the test shows that overall every client consumed about 122 seconds to finish the all operations. The workload of the master is 3443 operations per second. In 2003 Google's paper, they mentioned that the rate of operations send to their master is nearly 200 to 500 operations per second. This master can easily keep up with this rate. There is no bottleneck even for the workload of 3000 Ops/s.

b) ***Read Buffer Size***

When we are using different read buffer size then one client reads a 1GB region from a file. Figure 4 shows different buffer size and their read rate. The read rate reached its maximum when the size was around 1024KB, and after that becomes flat. So we choose 1024KB as the default value of the read buffer size.

Fig 6 Read Buffer Size [7]

c) ***Reads and Record Appends***

Because one switch is used to connect all our clients and servers machines, so to get the theoretical network limits, we run all clients as different threads on single client machine. So limit of network throughput of all clients is bound to 125MBps. When M clients read simultaneously from file system then each client reads a randomly selected 4 MB region from the 18 GB file.

It is repeated 256 times so that each client ends up reading the 1 GB of data. Figure shows the aggregate read rate for M clients. The limit peaks at an aggregate of 125 MB/s when client's 1 Gbps network interface gets saturated and Aggregate read rate reaches 90 MB/s, nearly 72% of 125 MB/s network limit. Sometimes most likely the efficiency of single client drops as the number of readers increases, because the probability that multiple readers simultaneously read from the same chunkserver increases. Figure shows the record append performance. M clients can append simultaneously to a single file. The performance is limited by aggregate bandwidth of links between the chunkservers and the clients. It is 125 MBps in given network topology. Aggregate append rate reaches 95 MB/s, nearly 75% of the 125 MB/s network limit. In another test, we run multiple clients on different machines.

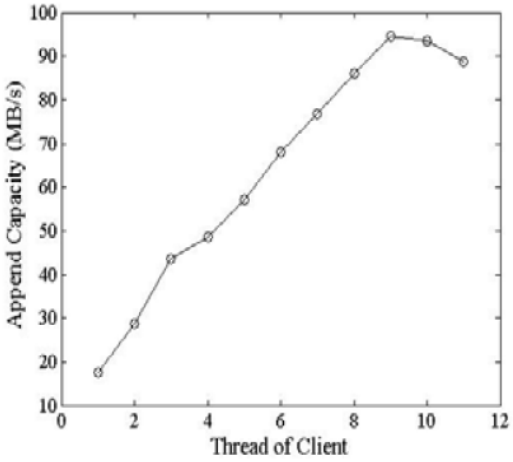

Fig 7 Aggregate Record Append Rate[7]

The result's shows that aggregate append rate can easily more than 380MB/s, which is a demonstration of expectation that out record append performance will not be drop due to the network limit of one chunkserver like in GFS. In contrast to this performance, the read and record append operations in the Google file system can reach 75% and 50% of the given theoretical limit, separately. So the record append performs much better. During the whole test, the rate of CPU of that client machine always maintained less than 5%, so there is no contest of CPU which may lead to the deviation of the results.

V. CONCLUSION

Although this system have same assumptions and same architectures with Google file system, but key design choice that the chunk size is variable, which is different from Google File System. Therefore, it lets this system to adopt different system interactions for the records append operation. The experiment results showed that this design

DOI 10.17148/IJARCCE.2015.4522

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

significantly improves the record append performance by 25%. We believe this design may apply to other similar data processing infrastructure. We find out that using the same system interaction for both record append and write is a limitation of GFS and restricts the possibility of digging for more better append performance, which led to our different points in the design space. We assume chunk size of file is variable and record append operation is based on chunk level due to which the aggregate record append performance is no longer limited by the network bandwidth of the chunkservers that store the last chunk of the file.

REFERENCES

[1] Alexandros Biliris, "An Efficient Database Storage Structure for Large Dynamic Objects", IIEEE Data Engineering Conference, Phoenix, Arizona, pp. 301-308, February 1992.

[2] An Oracle White Paper, "Hadoop and NoSQL Technologies and the Oracle Database", February 2011.

[3] Cattell, " Scalable sql and nosql data stores", ACM SIGMOD Record, 39(4), pp. 12–27,2010.

[4] Russom, " Big Data Analytics", TDWI Research, 2011.

[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "Google file system", 2003.

[6] Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach,Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber , Fay Chang, Jeffrey Dean , "Bigtable: A Distributed Storage System for Structured Data", OSDI 2006.

[7] Zhifeng YANG, Qichen TU, Kai FAN, Lei ZHU, Rishan CHEN, BoPENG, "Performance Gain with Variable Chunk Size in GFS-like File Systems", Journal of Computational Information Systems4:3 pp- 1077-1084, 2008.

[8] Sam Madden, "From Databases to Big Data", IEEE Computer Society, 2012.

DOI 10.17148/IJARCCE.2015.4522