Google File System Simulator

Pratima Kolan
Department of Computer Sciences
University of Wisconson-Madison
pratimak@cs.wisc.edu

Vinod Ramachandran
Department of Computer Sciences
University of Wisconson-Madison
vinorama@cs.wisc.edu

December 23, 2010

1 Abstract

Google File System(GFS) is a scalable distributed file system designed for large data-intensive applications. The filesystem has a complex architecture and undertanding its behavior when subject to various workloads is an interesting and challenging problem. In this project we made an attempt to solve this problem by simulating this filesystem and its components. We then studied the behavior and performance of the system when subject to various read workloads. We have simulated load balancing, GFS replication, GFS rebalancing and a disk failure for our experimentation and then observed how GFS performs in each case.

2 Introduction

The Google filesystem[1] is a scalable distributed filesystem designed for large data intensive applications. It is a filesystem designed to run on in-expensive commodity hardware and has good performance and fault tolerance. It caters to the storage needs of large data intensive applications used at Google and provides for several Terabytes of storage.It shares the same principles of earlier distributed filesystems, but differs in the sense that it has been designed for Google's applications workloads , which in most cases are sequential read and append workloads. Most of these workloads do not require existing locations to be re-written to.

This filesystem has a complex architecture and understanding the behavior of this architecture over various request workloads would be an interesting exercise. This is the main aim of our project as we try to understand the behavior of GFS over various read workloads.

We approached this problem by simulating the GFS architecture using discrete event based simulation. To do this we understood the GFS data flow mechanism and then simulated the various components of GFS like a client, master server and chunk servers. We then designed some experiments which were aimed at exploring load balancing, replication, re-balancing techniques across GFS chunk servers. We also simulated a disk failure and observed how the failure of a single disk can affect read performance. The reason we focussed on just read requests is because of the time constraints that we had. We had about six weeks to do the development and testing of this simulator. Simulating a read request and understanding its behavior over various workloads seemed like a well defined problem for this duration. We feel that a study of a write request and its behavior when subject to different workloads could be a good extension to this work.

Our simulator implements load balancing using a greedy strategy where read requests are distributed across all the chunk servers , this inturn leads to a greater effective data rate at the client which issues these requests. This is something that GFS does not do and is something worth thinking

about as it definitely improves read rates.

In the replication strategy we try to apply a dynamic chunk replication technique to obtain greater data rates.We increase the number of replicas for those chunks which are requested more frequently. GFS uses this technique to tackle hot spots in its system. Hot spots are chunk servers that receive many requests for a single popular file, like an executable file.

The rebalancing techique involves the movement of replicas from one disk to another and is done when the number of requests in a given chunk server increases beyond a certain point. This is manner in which standard GFS implements loadbalancing. It is slightly different from the load balancing that we try as we are using a simulator which takes a global snapshot of the number of requests being served by each chunk server, and uses this informaiton to route new requests to the chunk server with least load. Standard GFS also does rebalancing when it finds a given chunks server's disk space has fallen below the average free space in order to maximize disk space utilization and also request load is not uniform across the chunk servers.

Disk failures are very common in GFS as it uses commodity hardware. This is the main reason we felt it would be interesting to simulate such an event and see the performance hit that read requests would take.

We feel the work done in this project provides more insight into the behavior and performance of a distributed filesystem like GFS. The experimentation we provide can be treated as a preliminary step to a more broad range of tests that can be run on such a simulator. Buiding such a simulator also involves the challenge of ensuring the correctness of the results obtained, since everything is a simulation one must be careful in making sure he is doing the right thing and that a request data flow occurs in ths same was as it would occurs in the actual Google Filesystem.

The rest of this paper is layed out as follows, Section II talks about the GFS Architecture and Data Layout, Section III talks about related work, Section IV describes the basics of Discrete Event Based Simulation, Section V describes our implementation of the GFS simulator. Section VI talks about the experiments we performed and the observations made from these experiments. Finally Section VII concludes this paper and provides pointers for future work.

3 Google Filesystem

3.1 GFS Architecture

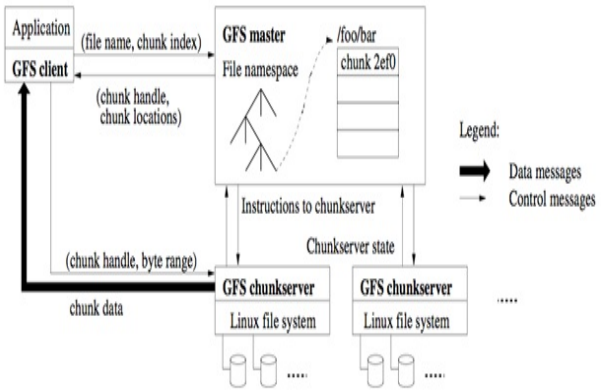


Figure 1: GFS Architecture

GFS consists of three main components, GFS Client, GFS Master and GFS Chunk server.

1. GFS Master Server: GFS Master Server keeps track of metadata of all the files in the filesystem

2. GFS Chunk Server: GFS chunk server stores certain chunks in its local disk which the client can access.

3. GFS Client: The GFS client implements filesystem API and interacts with the Master server and chunk servers to read and write data on behalf of an application. For any given request it first queries the master server for the location of the chunk servers that contain a copy of the data requested.

It further communicates with the nearest possible chunk server for the data. It caches the location of a chunk servers for a certain period of time inorder reduce the amount communication between the client and the Master server.

Files in GFS are divided into fixed size blocks called chunks. Each chunk has a unique id called chunk handler. Every chunk is a 64 MB block and is stored in a chunk server. The chunks are replicated across various chunk servers to provide for fault tolerance. The main reason that GFS has such a large chunk size is 1)to reduce the amount of metadata information requests from a client to the master server. For any reads and writes to the same chunk a client needs to query the master only once. 2) a larger chunk size helps the client to keep a persistent TCP connection with a chunk server and 3) it also helps in reducing the amount of metadata stored at the master server.

4 Related Work

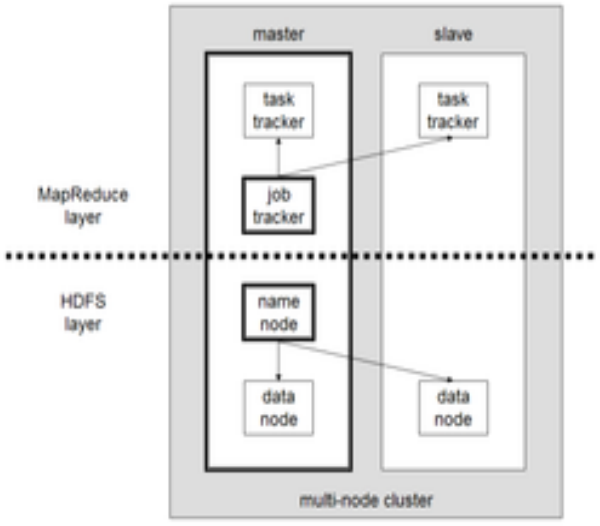


Figure 2. Hadoop Distributed File System

The Hadoop distributed filesystem[5] is a distributed file system written for hadoop. The inspiration for this filesystem came from the Google Filesystem as Google has not released its version of GFS for public use.Like GFS , HDFS has a single nameserver analogous to the GFS Master. It divides a file into 64 MB chunks and uses replication for fault tolerance. It has a miniimum replication of 3 chunks per chunk of data. There has been some work on the performance evaluation of HDFS when it is subject to various workloads. Shafer et al[2] , have come up with some interesting bottlenecks in the hadoop stack and HDFS. They have shown that architectural bottlenecks exist in the Hadoop implementation that results in inefficient HDFS usage due to delays in scheduling new MapReduce tasks. They also state that portability limitations prevent the Java implementation from exploring the features of the native platform.

The work in our project is different from this work. We are aiming to simulate the GFS itself and not use any open source implementation of its design. Our aim is not to find performance bottlenecks in a code base, rather we are trying to see if any design decisions or file system storage strategies could be suggested to improve GFS.

5 Discrete Event Based Simulation

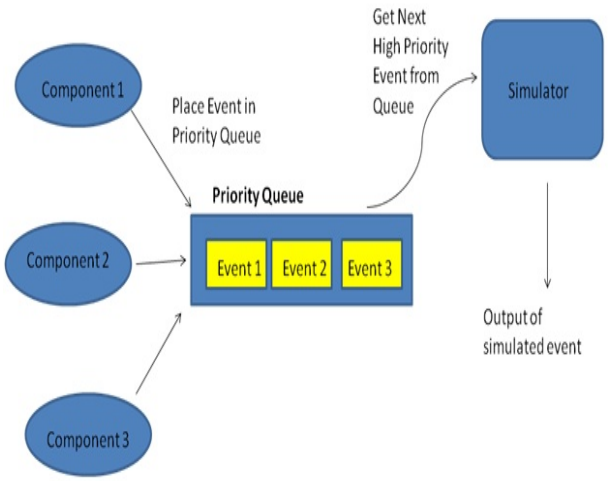


Figure 3: Discrete Event Based Simulation

Before we get into the details of how we simulated

GFS we would like to provide a small description of how a discrete event based simulation is done.
Discrete event based simulation[3,4] is a method of simulation in which the functioning of a system is represented as a series of events. Each operation in the system is represented by one event. The various components of the system place their events in a priority queue .A global simulator process extracts the event with the highest priority from the queue and then simulates it. Figure 3 above depicts this process where each component places its event in the priority queue and the simulator processes all of them one by one.

6 Google Filesystem Simulator

In this project we model the various component of Google Filesystem namely the GFS Client, GFS Master and the GFS chunk servers. We modeled a GFS chunk server as a network disk in order to keep things simple. We also have a network switch in our infrastructure. We have certain daemons running on the GFS components that periodically check for pending events to be servered in their respective queues and process the events from these queues if they are not empty.

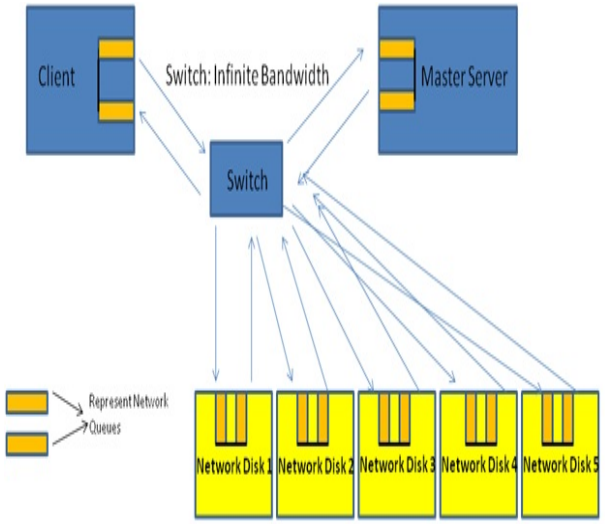


Figure 4: Google Filesystem Simulator Architecture

The following are the different events in our simulator:

- **Client Metadata Request:** This event represents a request from a client to the master server to fetch information regarding the location of a specific chunk, the client puts this request in a queue called the **ClientReqQueue**.
- **Client Data Request:** This event represents a request from a client to a chunk server for a specific chunk on that chunk server,the client puts this request in a queue called the **ClientReqQueue**
- **Client Network Simulator Request:** A daemon continously checks for a Client Request in the ClientReqQueue and processes the request if the queue is not empty. It then pushes the request into the SwitchReqQueue
- **Switch Request:** A daemon running on the switch periodically checks for any pendings requests in the SwitchReqQueue, when it finds a pending request it routes the request to either Master Server or Chunk Server depending on whether the request is for Metadata or Data.
- **Master Server Network Simulator Request:** This event pushes a client request for METADATA into the **MasterServerReqQueue**.
- **Master Server Request:** A daemon running at the Master Server pops a client request for METADATA from the MasterServerReqQueue , services it and initiates a Master Server Response by pushing a response into the **MasterServerResQueue**.
- **Master Server Response:** This event pops a response from the MasterServerResQueue and initiates a Master Server Network Simulator Reponse event.

- **Master Server Network Simulator Response:** This event pushes a Switch Response event into the **SwitchResQueue**.

- **Switch Response:** This event pops a response from the SwitchResQueue and initiates a Client Network Simulator Response event.

- **Client Network Simulator Response:** This event pushes a response from Master Server or Chunk Server into the **ClientResQueue**

- **Client Response:** This event pops a response from the ClientResQueue
  1. If the response is a METADATA response it initiates a Client Request event to a Chunk Server
  2. If the response is a DATA response it displays it to the client.

- **Chunk Server Network Simulator Request:** This is analogous to the Master Server Network Simulator Request event, this pushes a client request for DATA into the **ChunkServerReqQueue**.

- **Chunk Server Request:** A daemon running on the chunk server periodically checks for any pending data requests from the client,on finding a pending request it serves the request and initiates the Chunk Server Response event.

- **Chunk Server Response:** This event represents a response from the chunk server to a client along with the data for a specific chunk request from the client , it initiates a Chunk Server Network Simulator Reponse event

- **Chunk Server Network Simulator Response:** This event pushes a Switch Response event into the **SwitchResQueue**.

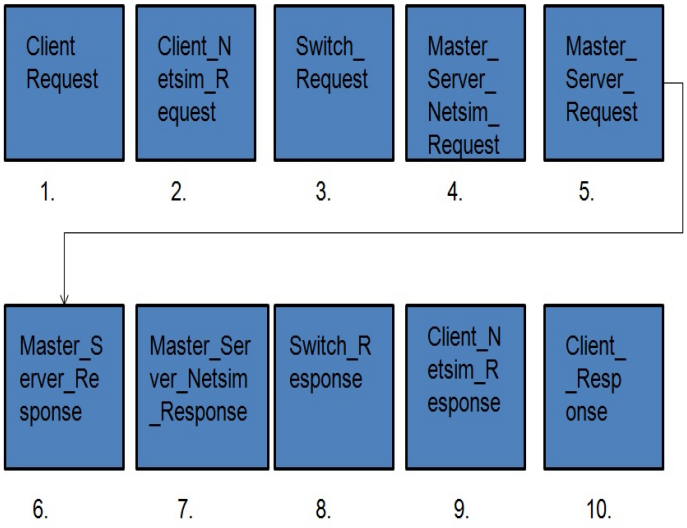


Figure 5: Client METADATA Request Flow

We simulate a read request for a client by simulating the meta-data requests between the client and the master server and the data requests between the client and the chunk servers. The main events in our simulation process for a client read are:

The flow for a Metadata Request is described in Figure 5. shown above.

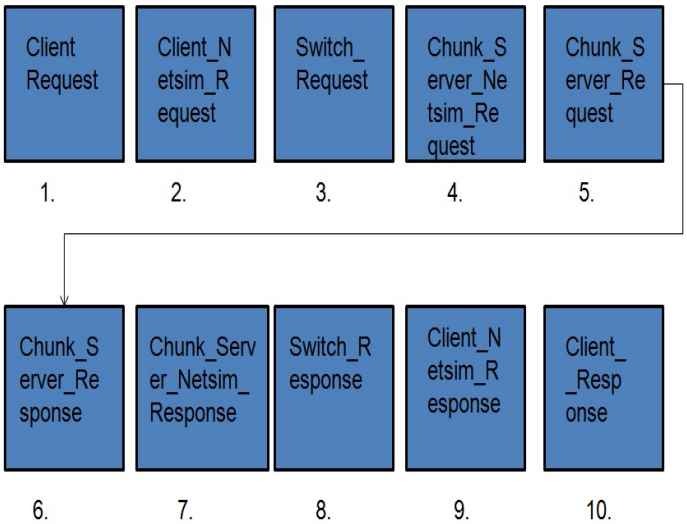


Figure 6: Client DATA Request Flow

The flow for a DATA Request is described in Figure 6. shown above.

7 Experimentation

Experimental Setup

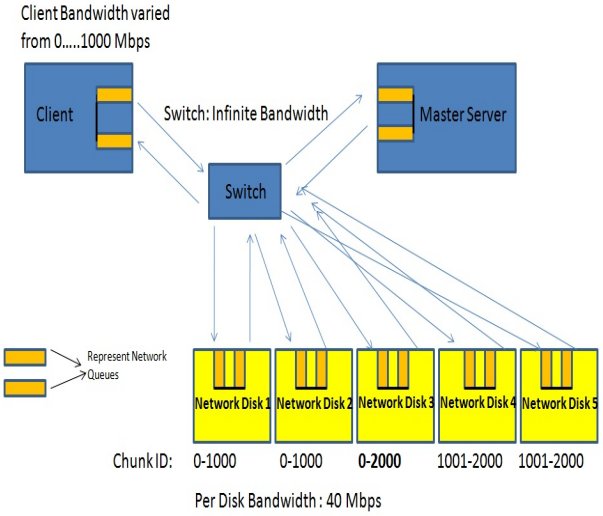


Figure 7: Experimental Setup

We are mainly focusing on experiments to measure the effective data rate obtained at a client for various read workloads in the Google Filesystem. The infrastructure described above was used to study the behavior of certain read workloads on Google Filesystem. Our experimental setup is described as follows:

We have one client, one switch, 1 master server and 5 chunk servers in our GFS setup.

1. We consider the switch and master server's bandwidth to be Infinity.

2. Each of the chunk servers has a bandwidth of 40 Mbps.

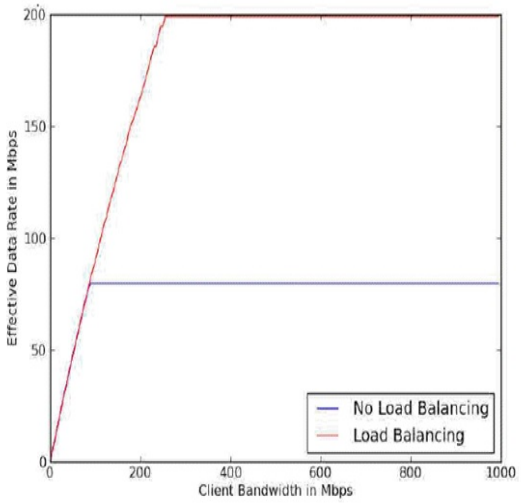
3. The distribution of the data across the five chunk servers is as follows:

Chunk Server 0: Chunk ID 0-1000
Chunk Server 1: Chunk ID 0-1000
Chunk Server 2: Chunk ID 0-2000
Chunk Server 3: Chunk ID 1001-2000
Chunk Server 4: Chunk ID 1001-2000

We chose such a data distribution as we felt it would bring out the best in our experimentation. We vary the client bandwidth from 0-1000 Mbps and measure the effective data rate at the client. We try four different experiments, a load balancing experiment, a replication experiment, a rebalancing experiment and a disk failure experiment. For each of these experiments we have provided graphs which plot the effective data rate at the client vs the client bandwidth.

7.1 Load Balancing In GFS

Given a data request for a chunk and locations at which this chunk is present, we route this request to that particular chunk server which has minimum pending request queue size. We compare this technique with a baseline solution where we dont have any load balancing. In this experiment we sent 8000 requests from the client and each request was a read request for a chunk with an id between 0-2000.



Graph 1: Effective data rate vs client bandwidth, Load Balancing Experiment

Our expectation is that the effective data rate at the client would be that of 5 disks(200 Mbps) in the load balancing case as compare to that of 2 disks(80Mbps) in the non-load balancing case. This expectation was reinforced by the observation we made after completing this experiment. Graph 1, above is a plot of the effective client data rate vs the client bandwidth and is a good representation of the results of this experiment.

7.2 Replication In GFS

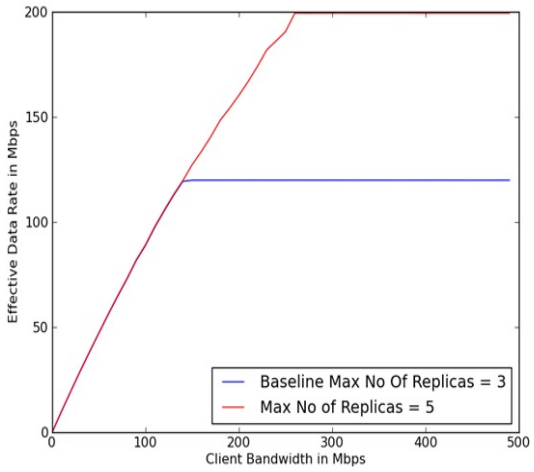
In this experiment we aim at studying the affect of dynamic chunk replication on GFS. Depending upon the frequency of requests for a particular chunk, we try to dynamically replicate chunks on to other chunk servers. Generally in GFS replication is done only when number of requests for a given chunk falls below some threshold. But, here we try to replicate a chunk depending upon number of requests for the chunk. Also we apply Load Balancing along with replication, as without Load Balancing there would be no point of doing any replication.

We define a term called Replication Factor which is a fraction < 1 .

We then consider this formula:

No of Replicas for a Chunk = (Replication Factor)*(No of requests for the Chunk)

We cap the number of replicas for a chunk by the total number of chunk servers available to us. After providing for dynamic replication, we consider a workload which queries for chunks placed on Disk 0, Disk 1 and Disk 2. The intuition behind such a workload is that as the frequency of the number requests for a chunk keeps increases, it would be replicated and would spread to all the five disks and such a workload would be useful in showing affect of replication.



Graph 2: Effective data rate vs client bandwidth, Replication Experiment

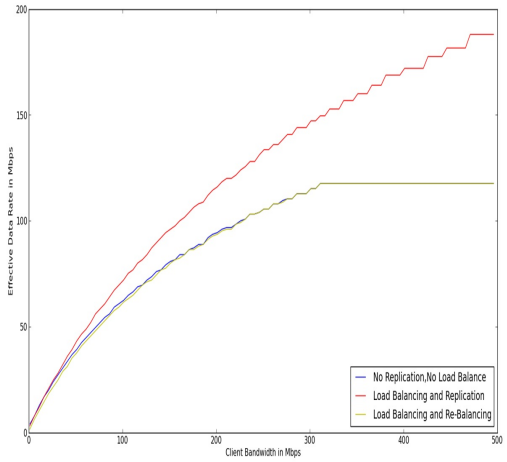
Our expectation is that the effective data rate with dynamic replication would be that of 5 disks while the effective data rate with no dynamic replication would be that of 3 disks. This expectation is re-enforced by the results of this experiment which are displayed in Graph 2 above. The effective data rate for the replication case is 200 Mbps, which is the total bandwidth provided by 5 disks and the effective data rate for the non-replication case is the bandwidth provided by that of 3 disks.

7.3 Re-Balancing in GFS

In this experiment we aim to study the performance improvement in GFS with respect to Re-Balancing. In this experiment, we periodically move the chunk which is requested more frequently on a chunk server to another chunk server which has minimum pending requests queue size and has minimum number of total queries served upto that time. We periodically repeat this process at every disk at regular intervals of time. The intervals should be very small as eager rebalancing does not lead to good distribution of load.

In this experiment we sent 8000 requests from the client and each request was a read re-

quest for a chunk with an id between 0-1000.



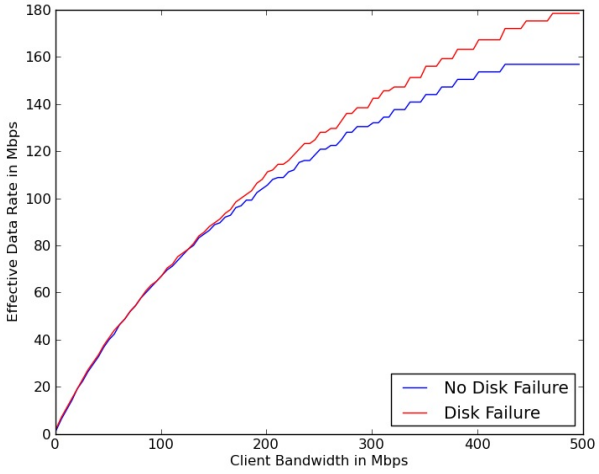
Graph 3: Effective data rate vs client bandwidth, Rebalancing Experiment

Our expectation is that the effective data rate at the client would be that of 5 disks in the rebalancing case as compared to that of 3 disks in the non-rebalancing case. This expectation was reinforced by the observation we made after completing this experiment. Graph 3, above is a plot of the effective client data rate vs the client bandwidth and is a good representation of the results of this experiment.

7.4 Disk Failure

In this experiment we aim at understanding the affects of a disk failure on GFS. We fail a disk after 1000ms from the start of the experiment and continue doing the experiment till 8000 requests are served. After the disk failure , the disk restarts again within a few milli seconds. It is assumed that all its data in the disk is lost during the failure. We then consider two cases as disk failure happens. The first case is when there is no replication and the second case is with replication. In this experiment we sent 8000 requests from the client and each request was a read request for a chunk with an id between 0-2000.

We expect in the no replication case effective data rate to be that of 4 disks(160Mbps) in the replication case as compared to that of 5 disks(200 Mbps) in the replication case. Our expectation is reinforced by the observation we made after completing the experiment. Graph 4 below is a plot of the effective client data rate vs client bandwidth and is a good representation of the results of this experiment.



Graph 4: Effective data rate vs client bandwidth, Rebalancing Experiment

8 Conclusion and Future Work

Google File System is a good file system for large data intensive applications. It has been designed to cater to the needs of various applications and workloads within Google. In this project we have tried to simulate the basic architecture and mechanism of this file system and then have studied the behavior of certain read workloads on it. We feel the load balancing idea discussed in this paper can complement the rebalancing strategy used by Google and can be considered as an optimization to Google File System. We feel that the simulation of a write request and the understanding of its behavior could be a nice extension to this project.

9 Acknowledgements

We thank Professor Remzi, for his support and motivation throughout the duration of this project. His guidance helped us divide the project into small components that we felt could be achieved in a period of 6 weeks.

References

[1] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T., "The Google file system.," In Proc. of the 19th ACM SOSP, pp. 29-43. , Dec. 2003.

[2] Shafer J , Rixner S , Cox,A.l, " The Hadoop Distributed Filesystem: Balancing Portability and Performance ," ISPASS , 122-133 ,March,2010.

[3] Thomas J. Schriber, Daniel T. Brunner, "Inside Discrete-Event Simulation Software: How it Works and Why It Matters.," Winter Simulation Conference , 77- 86 ,1998.

[4] Norm Matlo ." Introduction to Discrete-Event Simulation and the SimPy Language', February 2008

[5] D. Borthakur. The Hadoop Distributed File System:Architecture and Design. http://hadoop.apache.org/core/docs/current/hdfs design.pdf.