



---

# **Hercules File System**

## **A Scalable Fault Tolerant Distributed File System**

---

**Faraz Shaikh, Chaoyong Ragkhitwetsagul, Abdur Rehman Pathan**

Carnegie Mellon University

{fshaikh, cragkhit, apathan}@andrew.cmu.edu

Webpage: <http://www.andrew.cmu.edu/user/cragkhit/hercules/>

Mentor: Shobhit Dayal (sdayal@andrew.cmu.edu)

### **Abstract**

Current Distributed File Systems separate their servers into clusters of Metadata Servers (MDS) and Data Servers (DS). This separation of I/O access path into data and control paths allows parallel access to data from multiple clients to multiple data storage servers. Although metadata might constitute relatively small portion of the file system as compared to its overall size, metadata accesses might constitute significant percentage of overall I/O accesses thus making the scalability and performance of the MDS cluster of significant importance. Additionally, though the overall capacity of the DFS can be easily scaled by addition of additional data servers to the DS cluster, metadata exhibits a higher degree of interdependence, making the design of a scalable MDS cluster significantly challenging. We introduce the design of the Hercules File System (HFS), a distributed file system with scalable MDS cluster and scalable and fault-tolerant DS cluster. The Hercules File System allows Metadata and Data Servers to be dynamically added to the MDS cluster even after the initial setup time while the system is up and running without disrupting the normal operations carried out by the file system. The file system is also fault-tolerant and can serve clients in the events of failures of the DS and MDS. A Health Monitor is also designed which is a GUI tool that monitors the state of the servers of the File System and also gives the run-time visualization of operations requested by the clients.

## Contents:

1.	Introduction .....	4
2.	Project Requirements .....	6
3.	System Design.....	8
3.1	File System Client .....	8
3.2	File System Servers .....	9
3.3	Protocol RPC and Buffer Management.....	10
3.4	Health Monitor .....	11
4.	Application Programming Interface(s).....	13
4.1	FUSE (POSIX Calls support):.....	13
4.2	Client Library: .....	13
4.3	Mount a Client (Client Options).....	14
4.4	Unmount a Client .....	15
4.5	Server Options.....	15
4.6	Formatting the Hercules File System Servers .....	15
4.7	Configuration File .....	16
4.8	Update Notification to Client .....	16
4.9	Health Monitor Options.....	17
5.	Implementation.....	18
6.	Evaluation.....	21
7.	Demo/Use Sequence(s) .....	23
7.1	Basic File System Operations.....	23
7.2	Adding a New Node.....	23
7.3	Failure Resilience .....	23

7.4 Intermittent Failures .....	24
7.5 Health Monitor Functionality .....	24
8. Quantitative Performance Results .....	29
9. Project Status .....	31
10. Future Work .....	32
11. Conclusion.....	34
12. References .....	35

## 1. Introduction

A parallel file system is simply a component of a parallel I/O system that presents a hierarchical (file- and directory-based) view of data stored in the system. It includes a metadata server (MDS), which contains information about the data on the I/O nodes and a Data Server (DS) which contains the actual file data. Metadata is the information about a file—for example, its name, location, and owner. Some parallel file systems use a dedicated server for the MDS, while other parallel file systems distribute the functionality of the MDS across the I/O nodes. The main advantages a parallel file system can provide include a global name space, scalability, and the capability to distribute large files across multiple nodes.

Current examples of Parallel File Systems include PVFS, PVFS2, PanFS, Lustre and OGFS.

**PVFS (Parallel Virtual File System):** PVFS is jointly developed by the Parallel Architecture Research Laboratory at Clemson University and the Mathematics and Computer Science Division at Argonne National Laboratory. The metadata server in PVFS can be a dedicated node or one of the I/O nodes or clients. The node serving as the MDS runs a daemon called mgr, which manages the metadata for the files in the file system. In PVFS, data is distributed across multiple I/O nodes. The MDS provides information about how this data is distributed and maintains the locks on the distributed files for shared access. I/O nodes run a daemon called iod, which stores and retrieves files on local disks of the I/O nodes.

**Lustre:** Lustre is designed, developed and maintained by Cluster File Systems, Inc. It stores file system metadata on a cluster of MDSs and stores file data as objects on object storage targets (OSTs), which directly interface with object-based disks (OBDs). The MDSs maintain a transactional record of high-level file and file system changes. They support all file system namespace operations such as file lookups, file creation, and file and directory attribute manipulation—directing the actual I/O requests to OSTs, which manage storage that is physically located on underlying OBDs.

**PanFS:** The Panasas ActiveScale File System (PanFS) integrates an object-based clustered architecture to orchestrate file activity across the Storage Cluster and manage system performance. The file system virtualizes data across all StorageBlades, and presents a single, cache coherent unified namespace. PanFS simultaneously supports two high-performance data access modes: the DirectFlow data path and the NFS/CIFS data path.

**OGFS:** The OpenGFS uses a "Pool" driver to organize storage devices into a logical space. It stores data as blocks on this virtual block device. A locking subsystem, OmniLock, provides

the locking infrastructure necessary to ensure consistency. It also uses a virtual block device architecture, using LVM (Logical Volume Manager) underneath the GFS file system layer.

There are quite a few limitations to the above mentioned parallel file systems. To outline a few with reference to PVFS, the features missing currently are:-

- No caching of either data or metadata on the client side
- No scalability solution
  - Since its handle space is partitioned at format time, new servers cannot be added at run-time
- No scope for fault tolerance
- No write-sharing through mmap
- No support for Symbolic Links

To overcome some of these limitations, with the Hercules File System, we have attempted to achieve both scalability and fault-tolerance of meta-data servers and data servers. We also implement attribute caching on the client side.

## 2. Project Requirements

The Hercules File System was designed and implemented with the following requirements:-

- Support for POSIX Calls

The Hercules File System uses the File System in Userspace (FUSE) module to service requests from the clients. Thus, as FUSE provides POSIX compatibility, our file system would also support all POSIX calls providing the clients flexibility to run any POSIX compatible application in Linux.

- Data Server Fault Tolerance

High Availability is an important aspect of a Distributed System. File data is stored on the Data Servers in the Hercules File System. If any of the data servers fail, the file data would be lost. Hence, with active replication of the file data on a different data server, we would provide fault tolerant data servers. Only one failure of a data server can be tolerated by the Hercules File System. The clients would be unaware of such failures in the system.

- Meta-Data Server Fault Tolerance

The Meta-Data Servers in the Hercules File System store the attributes (meta-data) of a file. Hence, these servers must be highly available to serve clients. With MySQL Master-Slave replication, each MDS would have a backup MDS which would operate in case of a failure of the master MDS. The clients would be unaware of such failures in the system.

- Data Server Scalability

File data is striped across data servers. This provides higher read bandwidth. When the load on the data servers increases, addition of data servers must be supported to help reduce the workload and improve performance of the data servers. The Hercules File System would support such upward scalability of the Data Servers. This operation would be completely transparent to the clients using the file system, and would not require unmounting and remounting.

- Meta-Data Server Scalability

There would generally exist only one meta-data server (and its backup) in the initial configuration of the Hercules File System. But, as the number of clients would increase, as well as the namespace would increase, there is a need to add additional meta-data servers. The Hercules File System would support such upward scalability of the Meta-Data Servers. This operation would be completely transparent to the clients using the file system, and would not require unmounting and remounting.

- Run-Time Monitoring GUI Tool

Status of every machine in the system can be monitored in real time using the Health Monitor. It can facilitate a system administrator in system monitoring. It also benefits the system developer because it can show the activities happening on the servers at real-time. The application includes other interesting features, such as email notification, and SSH terminal. The application was written in Java, so it is platform-independent. This means that it can be run anywhere on any platform.

### 3. System Design

The system design consists of four main components as follows:-

#### 3.1 File System Client

This component has the implementation of the file-systems callbacks issued by fuse. We keep our implementation as agnostic to fuse as possible, by building a library that captures the eccentricities of our file systems. This would allow the file system to be ported to different client operating systems later.

As shown in fig. 3.1.1, the FUSE kernel module and the FUSE library communicate via a special file descriptor which is obtained by opening `/dev/fuse`. This file can be opened multiple times, and the obtained file descriptor is passed to the mount syscall, to match up the descriptor with the mounted file system, the Hercules File System.

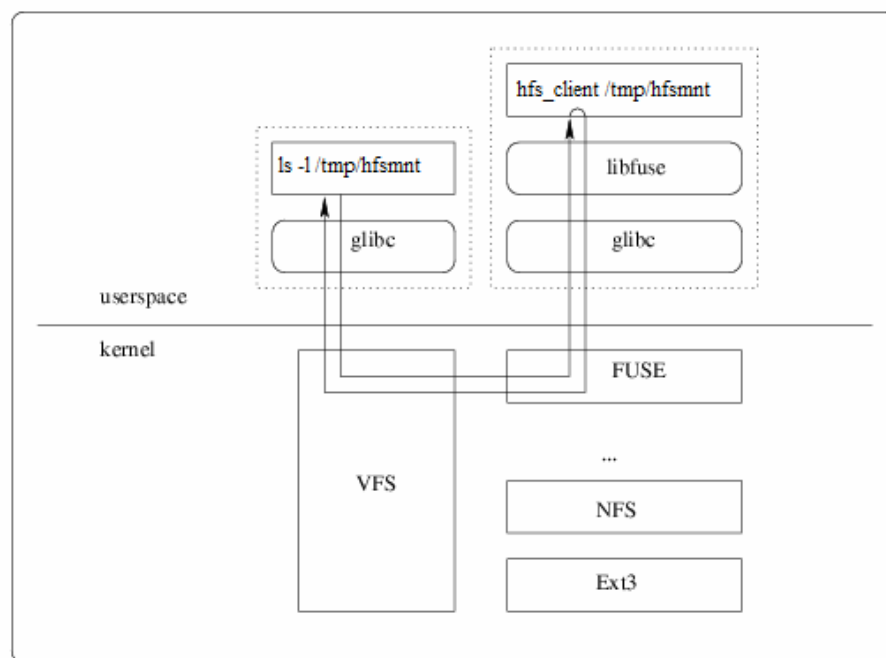


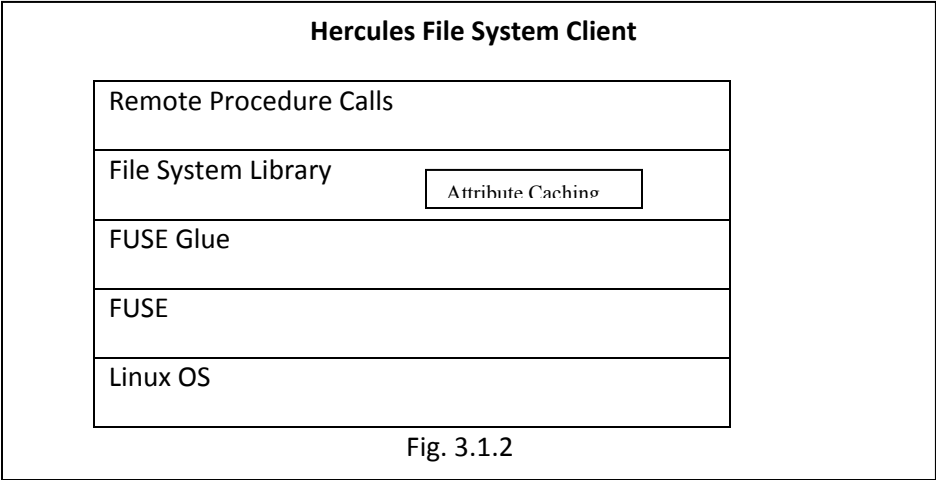
Figure 3.1.1

The Hercules File System Client would receive calls reflected by the FUSE module and then pass it on to the client library which communicates the file system servers using the protocol



RPC. The client additionally would be designed to cache the file attributes after an open() system call.

An abstract view of the Client architecture is shown in fig. 3.1.2

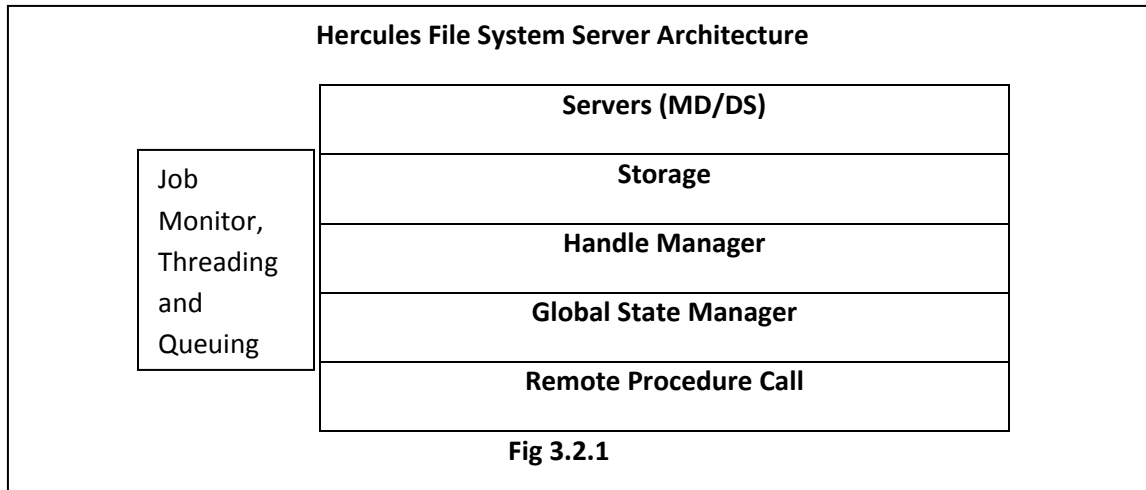


**3.2 File System Servers**

There are two types of servers in the Hercules File System – the meta-data server and the data server.

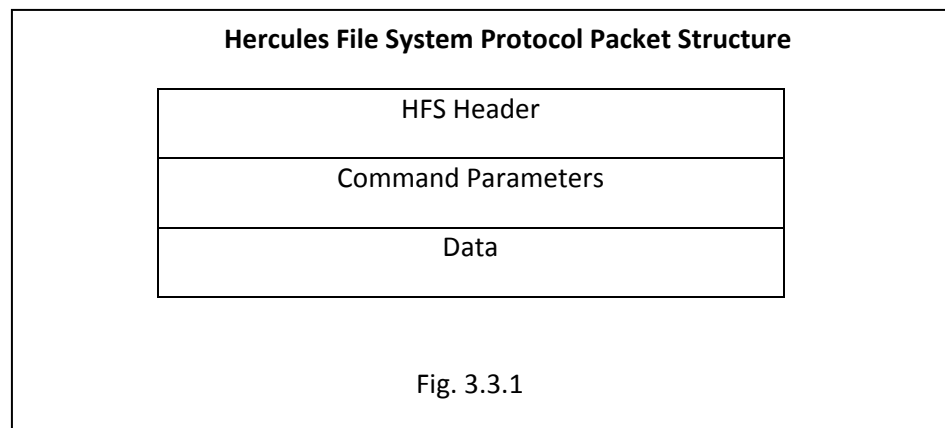
Both the servers would be built on the same design and would share the same code binary. A state machine is designed on the server to accept incoming requests. On the server, there is a thread pool per component. The activities to be performed by these components would be servicing the clients, polling, accessing the disk etc. An inbound thread polls continuously for client requests and on receiving a request, puts it on the appropriate queue i.e. As requests come in to the metadata server they will be queued in the server queue and then will be later dispatched to threads for execution. After the request is serviced by another thread, it puts it on the outbound queue. The outbound queue gets the request which was serviced and sends it back to the respective client.

An abstract view of the Hercules File System servers is as shown in fig. 3.2.1.



### 3.3 Protocol RPC and Buffer Management

The File System Client-Server Protocol consists mainly of a Header, Command Parameters and Data as shown in Fig. 3.3.1



A library would be written to encode and decode these RPC packets, given the respective command. This component takes care of structure packing/unpacking and handling the byte-ordering issues. The library would also contain functions, which would retrieve the command size, allocate data buffer depending on the type of the command. Thus, once this protocol buffer library is developed, it would be extremely easy to add a new command into the system.

Thus, adding a new command into the system would just mean adding it to the list of existing commands, adding its structure and listing it in the size retrieval function.

### 3.4 Health Monitor

The Health Monitor composes of many packages as shown in Fig 3.4.1.

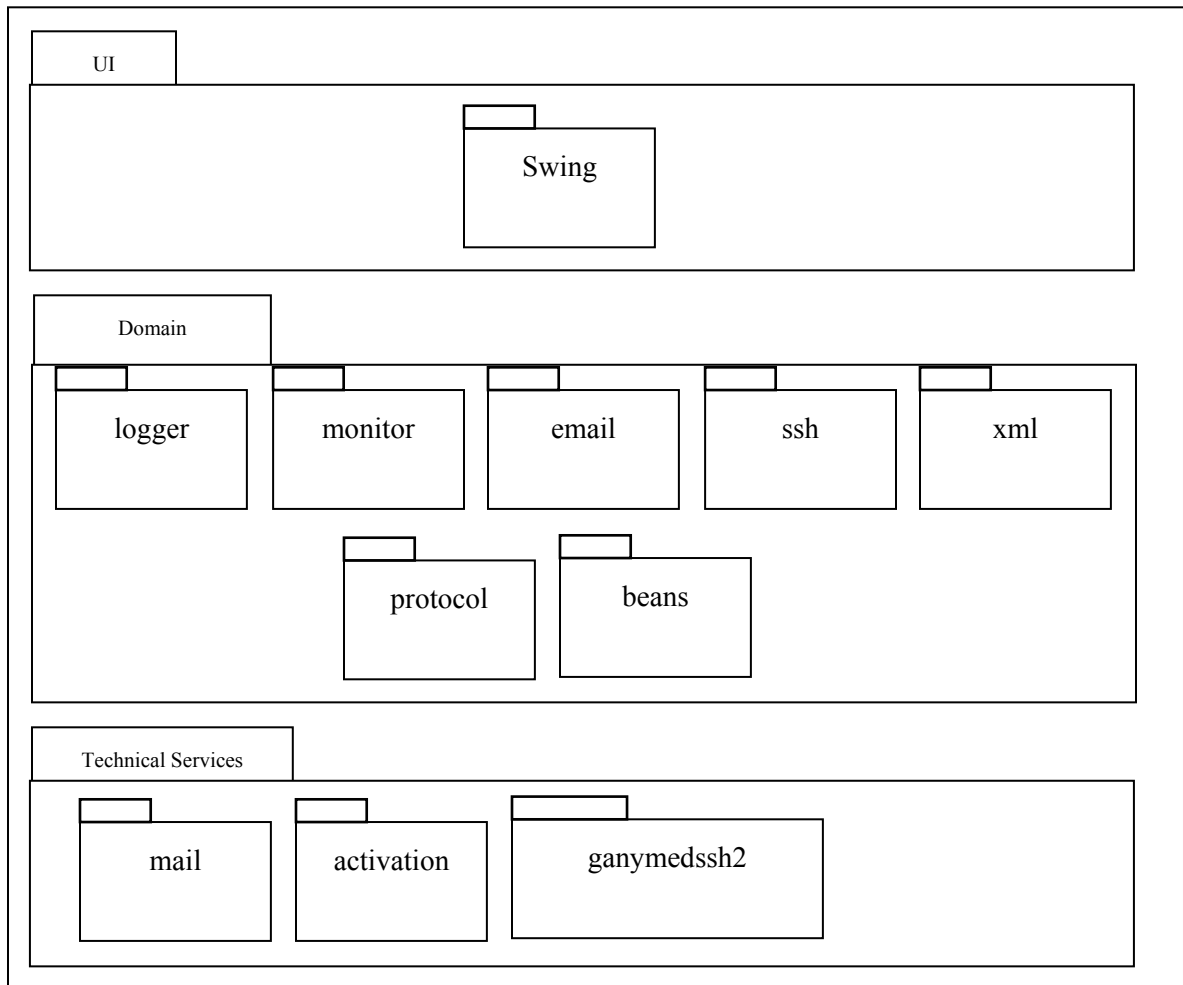


Figure 3.4.1 package diagram

The user interface was written in Swing. The domain package includes all the classes for creating log files, creating monitoring threads, sending email notification, create SSH terminal, reading XML setting file, converting from Java class to C struct, and data class (beans). The Health Monitor starts by connect to the root MDS to get all the configurations of all servers in the system. When it starts monitoring, it keeps sending ping packets to all the

machines to check alive. The ping replied packet contains the information of the latest activities happening on the servers. Then it converts all the data back in Java class format, and display on the screen.

## 4. Application Programming Interface(s)

The various application programming interfaces used in the Hercules File System are as follows:-

### 4.1 FUSE (POSIX Calls support):

```
static int clnt_fuse_getattr(const char *path, struct stat *stbuf)

static int clnt_fuse_open(const char *path, struct fuse_file_info *fi)

static int clnt_fuse_readdir(const char *path, void *buf, fuse_fill_dir_t filler, off_t offset, struct
fuse_file_info *fi)

static int clnt_fuse_mknod(const char *path, mode_t mode, dev_t rdev)

static int clnt_fuse_mkdir(const char *path, mode_t mode)

static int clnt_fuse_read(const char *path, char *buf, size_t size, off_t offset, struct
fuse_file_info *fi)

static int clnt_fuse_write(const char *path, const char *buf, size_t size, off_t offset, struct
fuse_file_info *fi)

void * clnt_fuse_init (struct fuse_conn_info *conn)
```

### 4.2 Client Library:

```
//- data -path ops -//

HFS_STATUS hfs_getattr(const char *ppath, struct stat *stat_buff);

HFS_STATUS hfs_read(PHFS_CLIENT_FILE_HDL pClientFileHdl, char *buffer,
off_t offset, IN OUT size_t *size);

HFS_STATUS hfs_write(PHFS_CLIENT_FILE_HDL pClientFileHdl, const char *buffer,
off_t offset, IN OUT size_t *size);
```

```

// Name space ops //

HFS_STATUS hfs_namei(char *name, PHFS_CLIENT_FILE_HDL pHfsClienthdl);

HFS_STATUS hfs_getlAttr(PHFS_CLIENT_FILE_HDL pHfsClientHdl, PHFS_IATTR
pHfsIAttr);

HFS_STATUS hfs_getlSize(PHFS_IATTR pHfsIAttr);

HFS_STATUS hfs_setlAttr(PHFS_CLIENT_FILE_HDL pHfsClientHdl, PHFS_IATTR
pHfsAttr, __u32 attrMask);

HFS_STATUS hfs_readdir(PHFS_CLIENT_FILE_HDL pClientFileHdl, off_t          offset,
PHFS_QUEUE_ITEM      *ppRetQueueItem);

// - Creation ops -//

HFS_STATUS hfs_create (char *ppath, mode_t mode, __u32 fs_dev_t,
PHFS_CLIENT_FILE_HDL pClientFileHdl);

HFS_STATUS hfs_create_alloc_data_handle(PHFS_IATTR piAttr);

HFS_STATUS hfs_create_alloc_meta_data_handles(PHFS_IATTR piAttr,
PHFS_CLIENT_FILE_HDL  pClientFileHdl);

//- Creation of a new directory entry. The namespace is distributed here -//

HFS_STATUS hfs_add_dirent(PHFS_CLIENT_FILE_HDL parentFileHdl,
PHFS_CLIENT_FILE_HDL childFileHdl, PHFS_DIRENT      pDirent);

```

### ***4.3 Mount a Client (Client Options)***

The client executable is executed in the following manner:-

```
$HFS/bin/hfs_client <mount_point> ROOT_MDS_IP_ADDR ROOT_MDS_PORT <logfile>
```

The mount\_point is a directory on the local file system.

The ROOT\_MDS\_IP\_ADDR and ROOT\_MDS\_PORT specify the IP address and Port of the root MDS from where the mapping of the servers is retrieved for the client to connect to.

The logfile would log all operations of the client.

#### ***4.4 Unmount a Client***

Unmounting a client is as simple as unmounting any other file system in Linux

```
umount <mount_point>
```

#### ***4.5 Server Options***

The Hercules File System Server executable begins with the following options:-

```
$HFS/bin/hfs_server <config_file> <server_id> <mds|ds> <max_no_of_clients_supported>
```

The config file specifies the number of Meta-Data Servers and Data Servers with their respective information such as IP Address, Port, Data Store Path, Log File etc.

#### ***4.6 Formatting the Hercules File System Servers***

The Hercules File System servers can be formatted using the following command:-

```
$HFS/bin/mkfs.hfs <config_file> <server_id> <mds|ds>
```

For the Meta-Data Servers, this creates a new database with a directory entry for root and namespace information for '.' and '..'

For the Data Servers, the base file is created in the data store path specified in the configuration file which is used for data handle management.

#### ***4.7 Configuration File***

Each server begins taking a configuration file as an input parameter. A sample configuration file is shown below:-

<MDS>

mdscount=1

localhost.localdomain 127.0.0.1      10000 TCP    /hercules/ds0 /hercules/log0.log

<DS>

dscount=4

localhost.localdomain 127.0.0.1      8090   TCP    /hercules/ds0 /hercules/log0.log4

localhost.localdomain 127.0.0.1      8091   TCP    /hercules/ds1 /hercules/log1.log

localhost.localdomain 127.0.0.1      8092   TCP    /hercules/ds2 /hercules/log2.log

localhost.localdomain 127.0.0.1      8093   TCP    /hercules/ds3 /hercules/log3.log

#### ***4.8 Update Notification to Client***

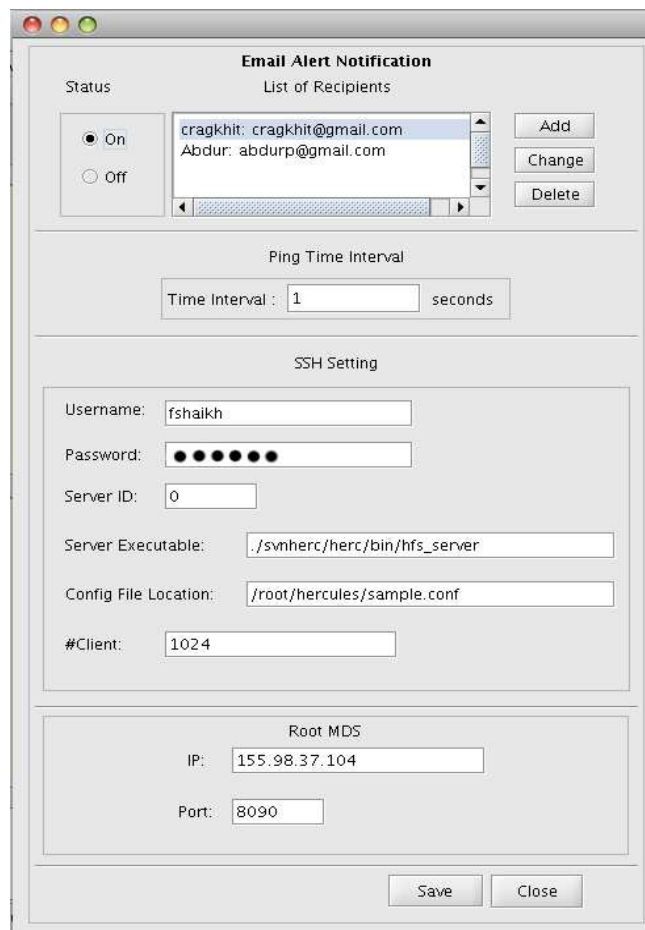
On addition of either a Meta-Data Server or a Data- Server, an executable is executed which sends a signal to a running server on that machine to notify the clients to update its file system configuration.

\$HFS/bin/updatehfsconfig

This must be executed on an existing server containing the list of all the connected clients.



## 4.9 Health Monitor Options



The screenshot shows a window titled "Email Alert Notification" with several sections for configuration:

- Status:** Radio buttons for "On" (selected) and "Off".
- List of Recipients:** A list box containing "cragkhit: cragkhit@gmail.com" and "Abdur: abdurp@gmail.com". To the right are "Add", "Change", and "Delete" buttons.
- Ping Time Interval:** A text field showing "Time Interval : 1" followed by "seconds".
- SSH Setting:** A group box containing:
  - Username: "fshaikh"
  - Password: masked with dots
  - Server ID: "0"
  - Server Executable: "./svnherc/herc/bin/hfs\_server"
  - Config File Location: "/root/hercules/sample.conf"
  - #Client: "1024"
- Root MDS:** A group box containing:
  - IP: "155.98.37.104"
  - Port: "8090"

At the bottom are "Save" and "Close" buttons.

The setting panel of the Health Monitor lets a user to do the followings:

1. Turn on/off the email notification
2. Add/remove/update receivers of the email
3. Ping time interval
4. SSH setting (username, password, server ID, etc.)
5. Root MDS IP
6. Root MDS port

## 5. Implementation

Queuing and Threading Model:

Both the servers and clients are multithreaded processes. As requests come in to the metadata server they are queued in the server queue and then will be later dispatched to threads for execution. This was implemented as a basic thread pool manager with service queues.

Update Notification to the Client:

On change in the file system configuration, an executable is executed which signals the server process. The server process then sends an unsolicited command back to the client, notifying it to update its configuration. The client then starts a transaction to update as follows:-

- TRANSACTION BEGIN
- FREEZE CLIENT
- WAIT FOR IN-FLIGHT COMMANDS
- UPDATE MAPPING
- CLOSE OLD CONNECTION
- TRANSACTION END

Thus, the client can adapt to the new file system configuration thereby supporting scalability.

File System Operations:

Create

The create file/directory performs several operations in the file system as follows:-

- Get\_Attr
- Namei
- Lookup

- Mknod
- Get Free Meta data Handles
- Get Free Data Handles

## Read

The Read operation gets data from both the data and meta-data servers. The data needs to be read off all stripes of the data servers.

The packets involved in a Read Operation are as follows:-

- Get\_Attr
- Namei
- Lookup
- Read

In case of failure of a data server, the client reads the data from its adjacent data server thereby providing fault-tolerant reads.

## Write

The Write operation writes data in stripes on the data servers. The data needs to be written on all stripes of the data servers.

The packets involved in a Read Operation are as follows:-

- Get\_attr
- Namei
- Lookup
- Read

The write operation performs active replication on the data servers. Each stripe is replicated on its adjacent data server. Thus, a client actively writes to both data servers.

Health Monitor:

Retrieve Machine Configurations

When the user press 'Get Machine Configuration from Server', the Health Monitor call `getExtentSize()` from the root MDS server. The returned value is the size of data being sent (not include the header part). After that, it send another command `getConfig()`, which returns the configurations of all the machines. The Health Monitor converts all the byte data into a Java instance, parse it, and display on the screen.

Start monitor

The Health Monitor starts monitoring the system, it starts sending ping packets to all the machines. The returned data are interpret (includes the activity light coding), and update the display panel of the MDS and the DS machines. The default ping time interval is 1 second.

Sending email notification

When the Health Monitor detects that a machine dies, it send an email to all the emails in the setting panel. This also includes the event when the machine becomes alive again.

Starting SSH terminal

A user can start a new SSH terminal from inside the Health Monitor. Select a machine from the display panel by clicking it, and click 'SSH' button. The SSH terminal will show up.

## 6. Evaluation

To evaluate the performance of our file system, we use machines from the EMU Lab test bed. The Hercules File System Team would like to thank Emu Lab for its support.

About EMU Lab: *Emulab* is a network testbed, giving researchers a wide range of environments in which to develop, debug, and evaluate their systems. The name Emulab refers both to a **facility** and to a **software system**. The primary Emulab installation is run by the Flux Group, part of the School of Computing at the University of Utah. There are also installations of the Emulab software at more than two dozen sites around the world, ranging from testbeds with a handful of nodes up to testbeds with hundreds of nodes. Emulab is widely used by computer science researchers in the fields of networking and distributed systems.

Our setup at EMU Lab consisted of the following components:-

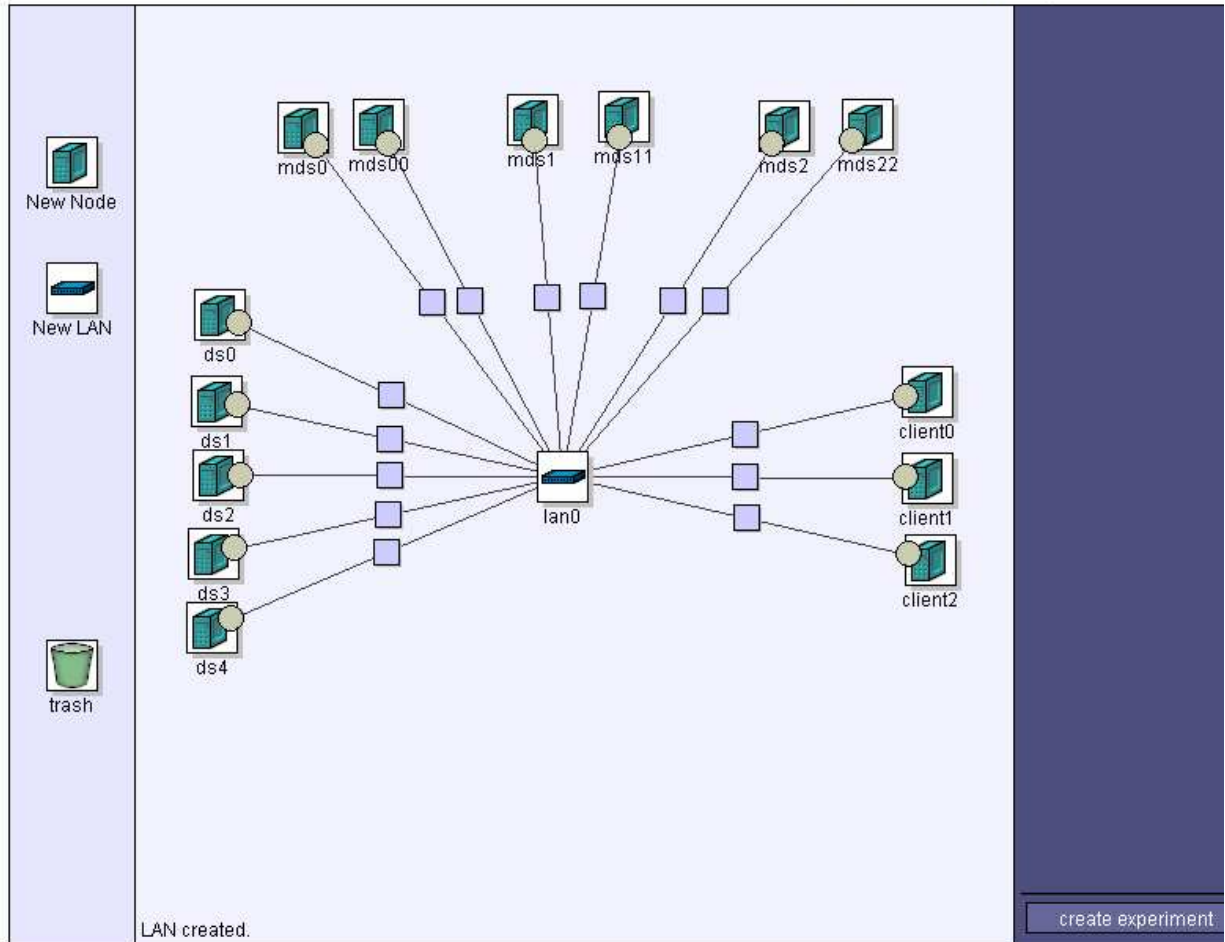
- 3 Primary Meta-Data Servers
- 3 Backup Meta-Data Servers
- 5 Data Servers
- 3 Clients
- LAN Connection among all nodes

To perform the experiments, we initially began with a configuration of only one meta-data server and 3 data-servers. Also, one client was used initially. We then added data servers and meta-data servers (along with their backups) into the system. The number of clients was also increased till three.

The total setup at EMU Lab is shown in fig. 6.1

## NetBuild

'fshaikh' Logged in.  
Wed Apr 16 9:51pm MDT



## **7. Demo/Use Sequence(s)**

### ***7.1 Basic File System Operations***

The demo included the working of the basic distributed file system client that supports basic file system operations such as Create (File), Open, Read, Write, Mount, Unmount, Create Directory. The file system supports applications compatibility as it provides support for POSIX calls. Audio and Video Files can be played by the traditional Linux players (e.g. mplayer, vlc etc.)

The operations performed during the demo were cp, cat, ls, ls -l, ls -lR etc.

### ***7.2 Adding a New Node***

A new node would be introduced in the system while the system is up and running. This would prove the concept of scalability. Our benchmark application was to demonstrate a multimedia application. The new node added could be either of the servers.

- **Meta-Data Server**

A new meta-data server was introduced into the system at run time. Its backup meta-data server (with the MySQL server replicated) was also be added. An update was sent to the client.

There was no downtime for the system. Continuous file system availability persisted.

The client was not mounted while this operation took place.

- **Data Server**

Data Servers were dynamically added to the system thereby increasing the handle space. There was no downtime for the system. Continuous file system availability persisted.

The client was not mounted while this operation took place.

### ***7.3 Failure Resilience***

The system also supported fail-stop failures. We do not handle network partitioning. Operations such as killing a meta-data server, killing data-servers were performed.

The multimedia application playing on the client side continued to play even in case of a failure of a data-server. In case of a failure of the meta-data server, the backup meta-data server was used by the client to service its requests. This was demonstrated successfully without affecting the client.

## 7.4 Intermittent Failures

To demonstrate an intermittent failure, a node (data-server) was removed from the whole system for a small amount of time after which it came back into the system. This was demonstrated while a multimedia application was playing at the client. The client was completely unaware of the intermittent failure.

## 7.5 Health Monitor Functionality

These are the screenshots of the working Health Monitor.

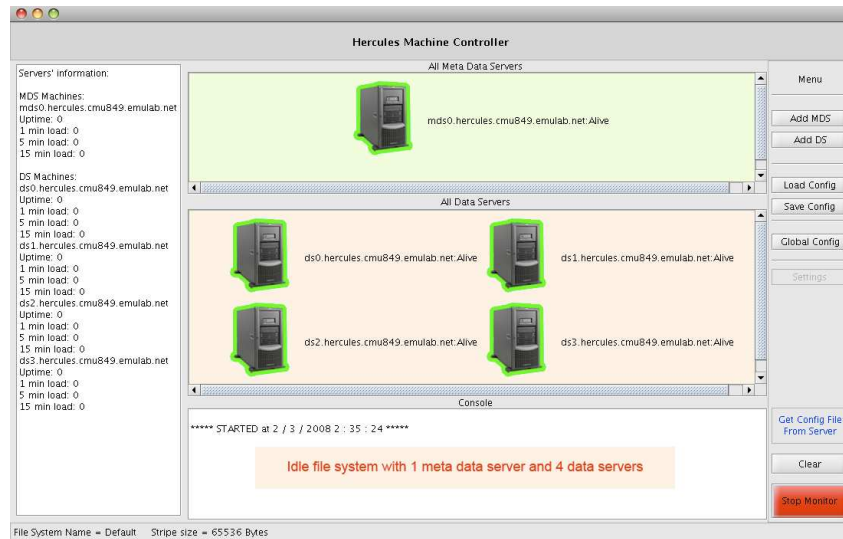


Figure 7.5.1 the system is in normal state. All machines are working.

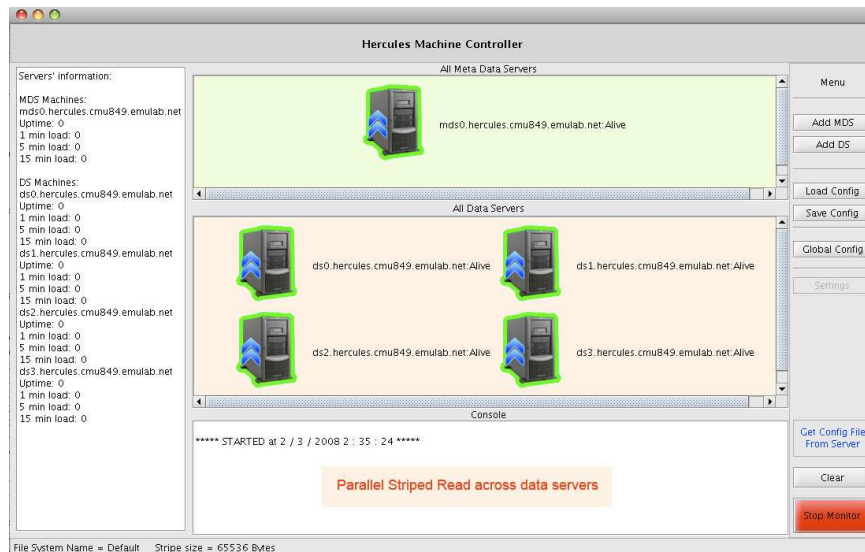




Figure 7.5.2 parallel striped read across all the data servers

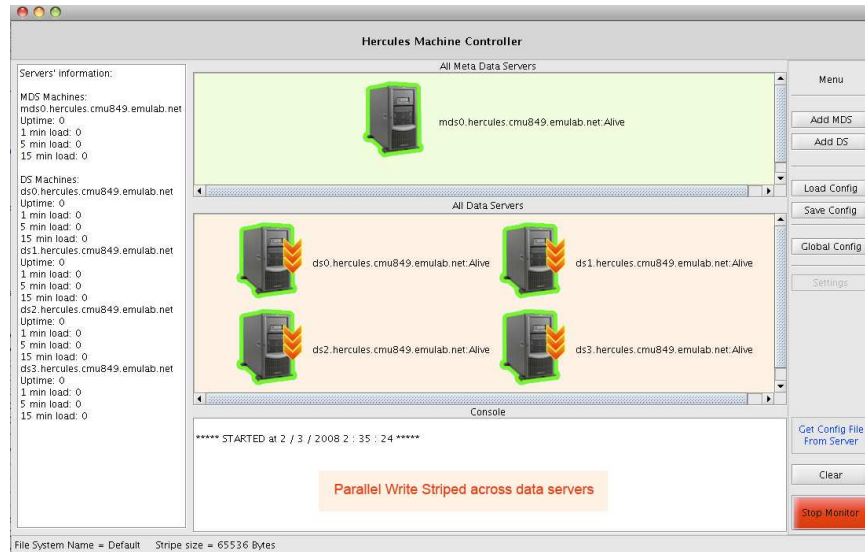


Figure 7.5.3 parallel write striped across all the data servers

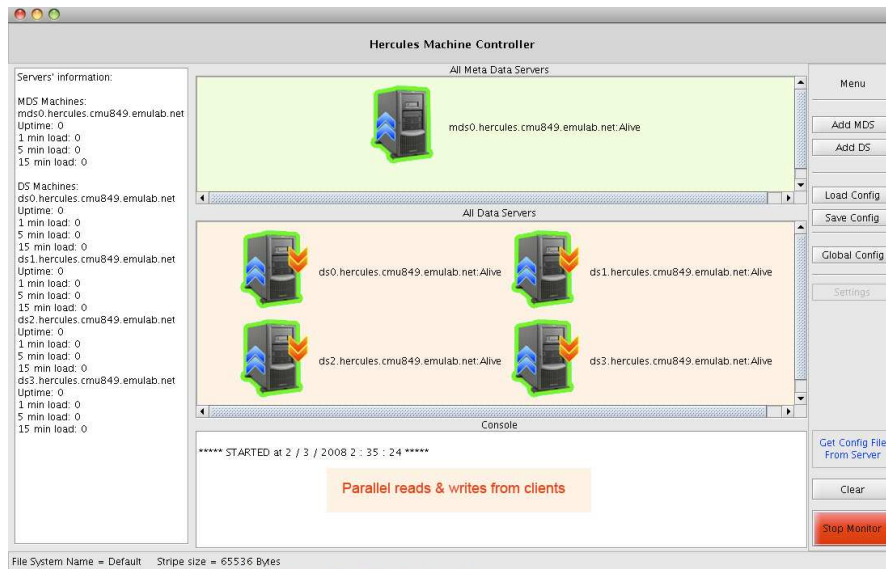


Figure 7.5.4 parallel reads & writes from clients

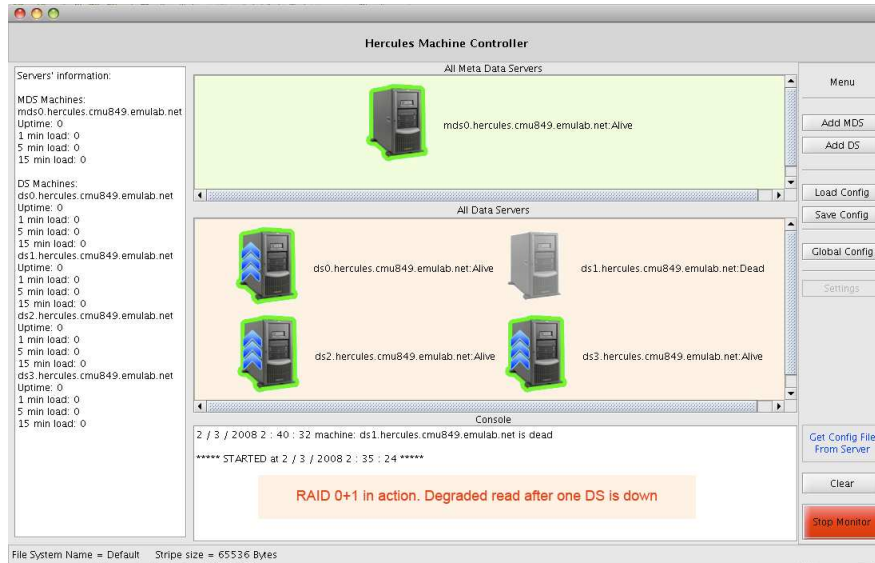


Figure 7.5.5 RAID 0+1 in action. Degraded read after one DS is down.

Email notification:

The Health Monitor notifies the users by sending an email when a machine is dead, or becomes alive.



Figure 7.5.6 the user receives email notification

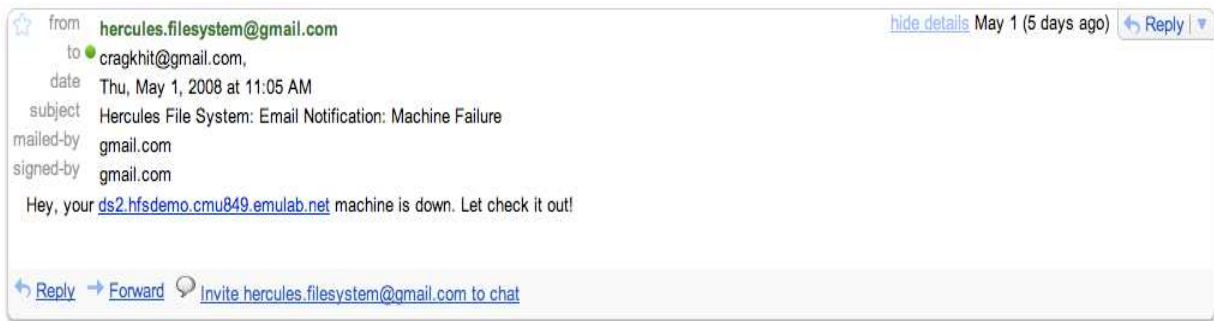
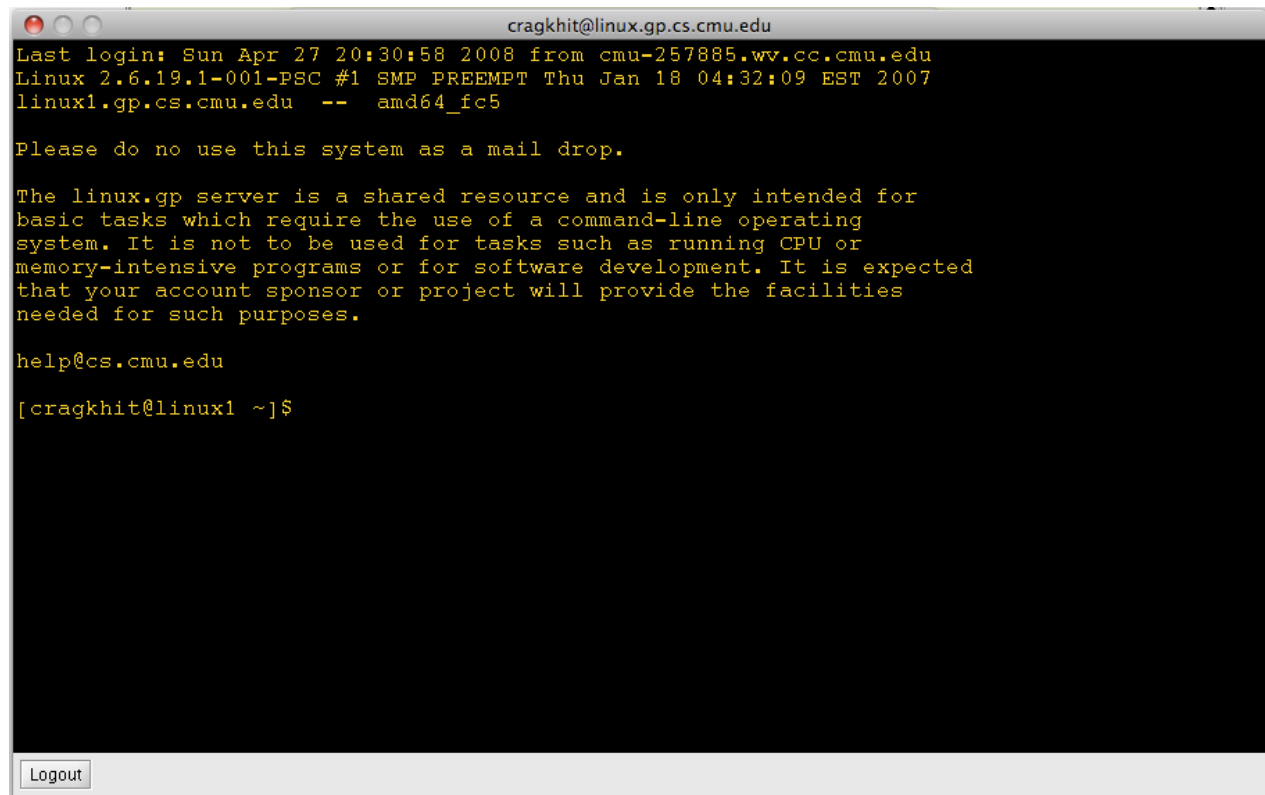


Figure 7.5.7 the email's content

## SSH Terminal:



```
cragkhit@linux.gp.cs.cmu.edu
Last login: Sun Apr 27 20:30:58 2008 from cmu-257885.wv.cc.cmu.edu
Linux 2.6.19.1-001-PSC #1 SMP PREEMPT Thu Jan 18 04:32:09 EST 2007
linux1.gp.cs.cmu.edu -- amd64_fc5

Please do no use this system as a mail drop.

The linux.gp server is a shared resource and is only intended for
basic tasks which require the use of a command-line operating
system. It is not to be used for tasks such as running CPU or
memory-intensive programs or for software development. It is expected
that your account sponsor or project will provide the facilities
needed for such purposes.

help@cs.cmu.edu

[cragkhit@linux1 ~]$
```

Logout

Figure 7.5.8 SSH terminal

## 8. Quantitative Performance Results

To evaluate the performance of our file system, we used the standard file system benchmarking tools like Postmark and IOZone. These tools are described in detail below:-

- PostMark

PostMark was designed to create a large pool of continually changing files and to measure the transaction rates for a workload approximating a large Internet electronic mail server. It generates an initial pool of random text files ranging in size from a configurable low bound to a configurable high bound. This file pool is of configurable size and can be located on any accessible file system. Once the pool has been created (also producing statistics on continuous small file creation performance), a specified number of transactions occur. Each transaction consists of a pair of smaller transactions:

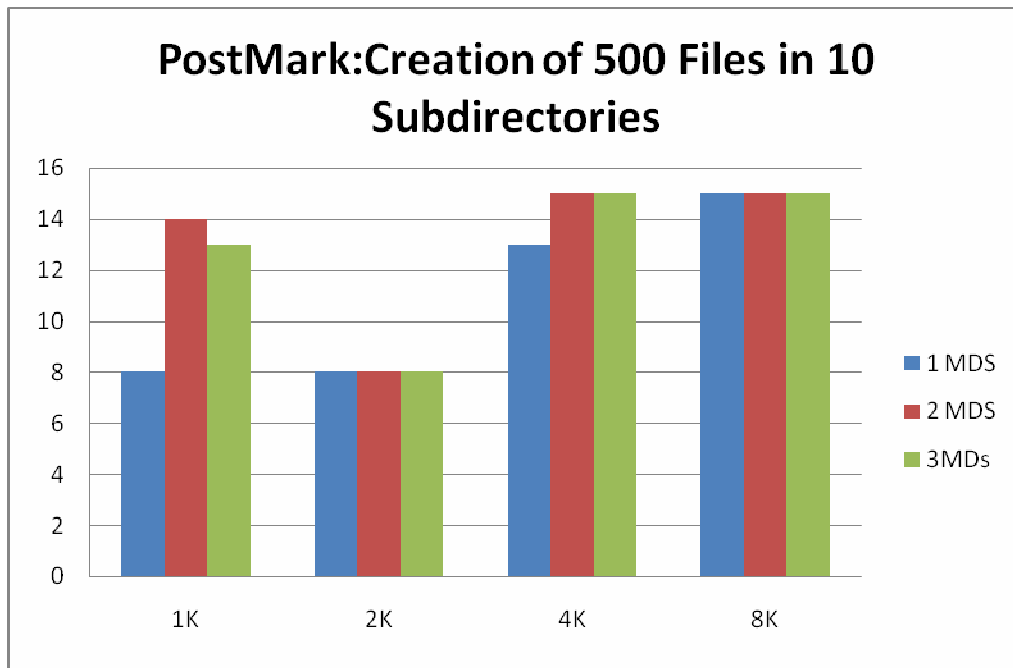
- Create file or Delete file
- Read file or Append file

- IOZone

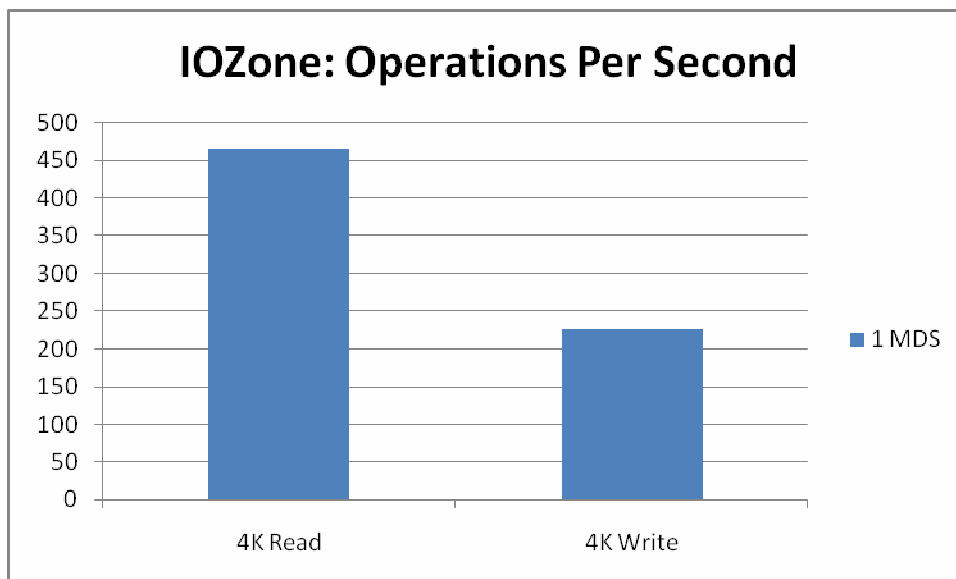
**IOZone** is a filesystem benchmark tool. The benchmark generates and measures a variety of file operations. Iozone has been ported to many machines and runs under many operating systems. Iozone is useful for performing a broad filesystem analysis of a vendor's computer platform. The benchmark tests file I/O performance for the following operations:

*Read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread, mmap, aio\_read, aio\_write*

It is POSIX and 64 bits compliant. It is also used for Distributed file server measurements (Cluster)



The Postmark benchmark is used to determine the number of meta-data operations, and from the graph it is evident, that we can create around 15 files per second.



The IOZone benchmark records the number of I/O Operations per second. Our file system with 4 data servers and 1 meta-data servers, and a stripe-size of 4kB gave around 460 reads per second and around 225 writes per second.

## 9. Project Status

The Hercules File System has been implemented as per the proposed design. It is currently operational in all of its claims. The Hercules File System final project demo was successful with all of its features.

The Hercules File System supports the following features:-

1. Working Distributed Parallel File System
2. POSIX Compatibility
3. Fault Tolerant Data Servers
4. Fault Tolerant Meta-Data Servers
5. Scalable Meta-Data Servers
6. Scalable Data-Servers
7. Health Monitor to give current state of servers and real-time visualization of operations
8. Attribute Caching on the Client Side

## 10. Future Work

Currently, the Hercules File System does not support some of the File System operations and coherency for data servers. The approach for solving these limitations is outlined below:-

- Rename Operation

The rename operation is a complex operation since the operation itself scans through multiple meta-data servers simultaneously. To accomplish this operation, would require ensuring distributed locking to maintain consistency of the namespace.

- Delete Operations (rm, rmdir)

The delete file and directory operations would require deletion of the name from the namespace, the inode and directory entries, as well as data files from the data servers. It would also require freeing both the meta-data handles and the data-handles.

- Coherency at Data Servers

In case of intermittent failures, the data server would need to sync up with its backup so as to get the latest copy of the data. This would require the data server silently restoring the missing data from its backup (while it had gone down).

- Optimizations at Data Servers

- Asynchronous I/O

Making the data server operations asynchronous would lead to a faster performance at the data server.

- Efficient Handle Management

The handle space can be efficiently managed and accessed with the help of a tree-based data structure. A simple implementation would be to implement the handles as a B-Tree thereby reducing the search times.

- Retrieve Modification Time from Data Servers (similar to getSize implementation)

With the current implementations, the size of a file is retrieved from all the data servers and then added to give the total size of a file. Similarly, the time can be retrieved from



all the data servers, and the latest of these times would be the modification time of the file.

## 11. Conclusion

The Hercules File System aimed at reducing the drawbacks of the current implementations of the parallel file system and was successful in doing so. Parallel file systems such as PVFS, Lustre are well suited for HPC cluster environments and have capabilities that fulfill critical I/O subsystem requirements. These file systems are designed to provide cluster nodes with shared access to data in parallel. They enable high performance by allowing system architects to use various storage technologies and high-speed interconnects. Parallel file systems also can scale well as an organization's storage needs grow. This can be achieved by the scalability of the Hercules File System. And by providing multiple paths to storage, Hercules File System can provide high availability for HPC clusters as well.

The file system state can be monitored at any time with the Health Monitor GUI Tool developed. Since the GUI tool also monitors operations at real-time, it is very useful to understand the working of a parallel file system.

The performance can be scaled well with the addition of data-servers and meta-data servers. There are still some optimizations remaining which would help us to increase the performance of the Hercules File System.

## 12. References

- “An overview of the Parallel Virtual File System” by Walter B. Ligon III, and Robert B. Ross
- “Standardizing Storage Clusters” by Garth Goodson, Sai Susarla, and Rahul Iyer, Network Appliance.
- “Intelligent Metadata Management for a Petabyte-scale File System” by Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Kristal T. Pollack.
- “The Global File System: A File System for Shared Disk Storage” by Steven R. Soltis, Grant M. Erickson, Kenneth W. Preslan, Matthew T. O’Keefe, and Thomas M. Ruwart.
- “Beowulf Cluster Computing with Linux, 2nd Edition” by William Gropp, Ewing Lusk and Thomas Sterling
- “Using Threads in Interactive Systems: A Case Study” by Carl Hauser, Christian Jacobi, Marvin Theimer, Brent Welch and Mark Weiser
- PVFS2: [www.pvfs.org](http://www.pvfs.org)
- PanFS: <http://www.panasas.com/panfs.html>
- Lustre: [http://wiki.lustre.org/index.php?title=Lustre\\_Documentation](http://wiki.lustre.org/index.php?title=Lustre_Documentation)
- OGFS: <http://opengfs.sourceforge.net/>
- “High-Performance, Reliable Secondary Storage” by Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson, 1994
- “A Case for Redundant Arrays of Inexpensive Disks (RAID)” by David A Patterson, Garth Gibson, and Randy H Katz
- FUSE: <http://fuse.sourceforge.net/>
- MySQL Server Replication: <http://dev.mysql.com/doc/refman/5.0/en/replication-howto.html>
- IOZone: <http://www.iozone.org/>

- Postmark: <http://www.freebsdsoftware.org/benchmarks/postmark.html>
- EMU Lab: [www.emulab.net](http://www.emulab.net)