Data Logistics and the CMS Analysis Model

Enabling Global Collaboration

Julie E. Managan
Vanderbilt University, Nashville, TN 37235

Abstract

The Compact Muon Solenoid Experiment (CMS) at the Large Hadron Collider (LHC) at CERN has brilliant prospects for uncovering new information about the physical structure of our universe. Soon physicists around the world will participate together in analyzing CMS data in search of new physics phenomena and the Higgs Boson. However, they face a significant problem: with 5 Petabytes of data needing distribution each year, how will physicists get the data they need? How and where will they be able to analyze it? Computing resources and scientists are scattered around the world, while CMS data exists in localized chunks. The CMS computing model only allows analysis of locally stored data, "tethering" analysis to storage. The Vanderbilt CMS team is actively working to solve this problem with the Research and Education Data Depot Network (REDDnet), a program run by Vanderbilt's Advanced Computing Center for Research and Education (ACCRE). I participated in this effort by testing data transfers into REDDnet via the gridFTP server, a File Transfer Protocol which incorporates an LHC Computing Grid security layer. I created a test suite which helped identify and solve a large number of problems with gridFTP. Once optimized, I achieved sustained throughputs of 700-800 Megabits per second (Mbps) over a 1 Gigabit per second (Gbps) connection, with remarkably few failures. GridFTP is the gateway between REDDnet and CMS, and my tests were designed to exercise and harden this important tool. My results support other indications that the REDDnet system will be a successful solution to the limitations of data-tethering in the CMS computing model.

1. Introduction

I will begin by presenting the general structure of the CMS computing model and the software used to analyze physics data. I will then describe the REDDnet system in greater detail, including the several data storage programs available to global users. Finally, I will present the test suite I designed to determine the capabilities and limits of Vanderbilt's gridFTP server in an end-user scenario. I successfully integrated scripts in several different languages to push thousands of files through the gridFTP server into REDDnet storage depots via the Logistical Distribution Network (LoDN). While many errors appeared throughout the testing period, most were related to server instabilities and were essential for rooting out problems with the gridFTP server, LoDN, or other programs. Exercising these instabilities and understanding how our tools interact with the computing infrastructure is a very important task for improving REDDnet.

2. CMS Data Analysis Model

Computing for the Compact Muon Solenoid is divided into Tiers and distributed globally. All signals that pass through the detector's triggering system are stored as raw data. Each collision that occurs in the detector is called an "event", and the raw data for an event contains all the information that was collected by the detector. CERN is the Tier-0 center where the raw data is grouped into datasets, stored, and backed up. CERN also conducts an initial reconstruction of all the collision events[1].

The raw data is then sent to seven Tier-1 centers located in France, Spain, Italy, Germany, Taiwan, the United Kingdom, and the United States (Fermilab), so that all datasets exist at CERN and one Tier-1 center. Each Tier-1 stores and backs up a copy of the raw and partially-reconstructed data it is given, and conducts more extensive reconstructions of the events. Tier-1 sites like Fermilab conduct analyses, but they are also responsible for supplying the data needs of the Tier-2 and 3 centers in their vicinity[2]. Tier-2 sites store the data needed by specific analysis subgroups. This is a simple demonstration of data-tethering, because in this model anyone conducting an analysis for one of these sub-groups would need to travel to the Tier-2 which has the data stored on site. Tier-3 centers are generally much smaller than Tier-2's, and only store data needed by local groups. ACCRE is a Tier-3 center for CMS, although their resources, such as the REDDnet system, resemble those of a Tier-2.

CMS data exists in a three-step hierarchy: RAW data, RECOnstructed events, and Analysis Object Data (AOD). RAW data contains all the data collected by the detector and stored at CERN. RAW data events are reconstructed and stored as RECO data at both CERN and the Tier-1 site for that dataset. At Tier-1 sites the RECO data is "distilled" into AOD, which is a data format compact enough to be useful for efficient physics analysis. Tier-2 and Tier-3 sites usually only receive RECO and AOD datasets to use for direct analysis[3].

CMS analysis is done using a software framework called CMS Soft-Ware, or CMSSW. New versions of CMSSW are released regularly, and every computing site is responsible for maintaining the current releases on their systems. Users then set up a project area for the release they want to use for analysis. Running CMSSW is, on the surface, quite simple, but in practice there is a steep learning curve! A configuration file written in python defines the analysis to be conducted, and the command 'cmsRun' is called to run the configuration file through CMSSW, producing a ROOT file with event data that can be viewed using the ROOT graphics capabilities (ROOT is CERN's Object-Oriented analysis package).

CMS analysis can be done both locally and remotely, through the CMS Remote Analysis Builder (CRAB). All CMS affiliated sites can register their Compute Elements (CE) and Storage Elements (SE) on the Open Science Grid (OSG), which makes them available for remote analysis. The VAMPIRE cluster at ACCRE is a compute element, and after several weeks of trials I was able to successfully run CRAB jobs using these resources. To use CRAB for a remote analysis, users create a CRAB configuration file which contains information regarding the dataset to be analyzed, the storage element where it is located, the CMSSW release, the local CMSSW configuration file, the desired compute element, and a destination for job output. CRAB uses grid tools to communicate with the computing infrastructure, and will farm out analysis jobs to sites that have the necessary datasets stored locally, returning the ROOT output to the

user. Once the CRAB configuration file is prepared, a simple set of commands is used to create, submit, check status, and get the output of jobs.

In practice, it is a difficult task to set up CRAB jobs to run at a new site, as I was trying to do at Vanderbilt. Initially, we did not have the correct connection protocol available and I had to manually edit each CRAB code to use the gridFTP server. CRAB will be very useful for physicists who lack local computational resources, but it is also a good demonstration of the data-tethering built into the CMS computing model. Users can only analyze data at sites that store the data locally. With CRAB, a user has tools to request computing time on distant machines, but unless they download and store their own data they cannot analyze it using their own resources, on their own time.

Physics Experiment Data Export (PhEDEx) is the data distribution system used by CMS. Users request datasets, and their requests must be approved by PhEDEx. It is a very hierarchical and structured system (based on the tiered computing model) that requires a lot of planning from users. This adds to the data-tethering problem: in the end, real creativity is going to happen accidentally, not when someone tries to schedule discovery. The goal of REDDnet is to solve this data-tethering problem so we can maximize user resources and accomplish the primary analysis goals of the experiment. It is very important to have a structured procedure for storing, distributing, and analyzing data, but the significant physics discoveries are going to happen on a smaller, less rigidly-scheduled scale and REDDnet it seeking to prepare for that.

3. REDDnet

REDDnet is a storage infrastructure system that "provides a large distributed storage facility for data intensive collaboration among the nation's researchers and educators in a wide variety of application areas"[4]. Vanderbilt founded this system to enable collaboration between geographically distant physicists who need to move data from one location to another or want to share large datasets for analysis projects. REDDnet uses Logistical Networking technology to move data around a physical network of storage depots housed at Vanderbilt, University of Florida, University of Michigan, California Institute of Technology, University of California Santa Barbara, the San Diego Supercomputer Center, Stephen F. Austin University, and CERN[5]. These depots are available to all REDDnet users through a number of logistical storage programs.

Three logistical storage programs have been developed for REDDnet use: L-store (ACCRE), LoRS, and LoDN (University of Tennessee at Knoxville (UTK)). UTK's Logistical Computing and Internetworking Lab (LoCI) developed the Internet Backplane Protocol (IBP) and Logistical Backbone (L-bone) which are the foundation of the REDDnet storage system. Their Logistical Runtime System (LoRS), a toolkit with a C-API, is designed to transfer data and process exnodes[6]. When a file is uploaded using logistical networking, information about the locations of the individual blocks is stored in a file called an exnode, which is read by the networking software to reconstruct the file for download. LoRS, like all of the logistical storage programs, has command line tools which allow the user to manually transfer data onto REDDnet depots. LoRS installation is also required to run ROOT.

L-store (short for logistical storage) is a java system that was developed at ACCRE for managing exnodes and communicating with IBP. The L-store command line tools are easier to use than the LoRS tools, and for a short time I ran a test which used the L-store program to transfer files from the VAMPIRE cluster in and out of REDDnet.

UTK's Logistical Distribution Network (LoDN) has garnered the most attention in our group over the past year. We usese the OSG Globus toolkit to upload and download files into LoDN, which distributes three copies of each file to user-defined depots. This process is known as "warming" and is done regularly to maintain the availability of the file. LoDN uses the LoRS toolkit to manipulate exnodes, extend leases, or delete data. The Globus uberftp toolkit is available to users to access the file structure and move their files. Dr. Daniel Engh has developed a plugin to the ROOT program which allows users to read files directly into their ROOT session by calling the exnode with a lodn:// protocol.

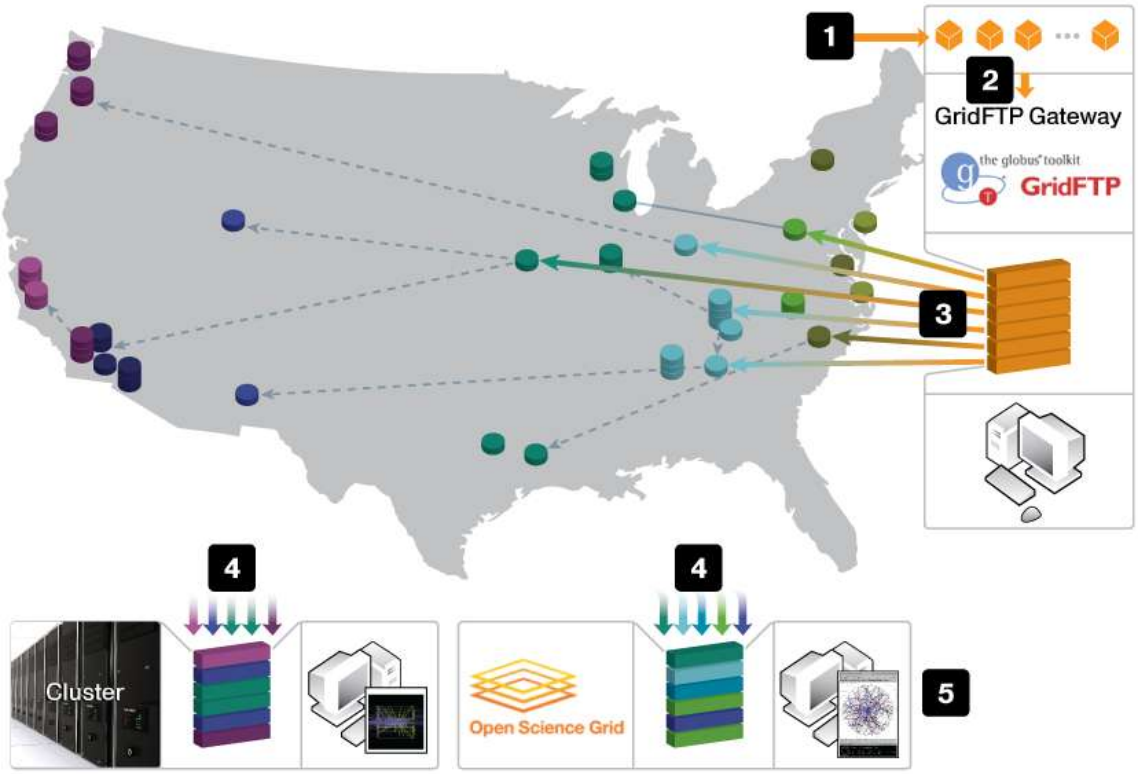


Figure 1: "Un-tethering CMS analysis through improved data logistics … (1) CMS computational resources produce CMS datasets. (2) These datasets are uploaded via gridFTP into REDDnet and are then (3) replicated and striped across multiple, geographically distributed storage depots. Replication strategies are designed to match usage, and more copies are made of data in greater demand. (4) CMS user analysis jobs stream data directly from REDDnet. Dynamic selection of depots optimizes streaming. (5) Users can access this data using local systems, grid resources, or other shared resources, and access is transparent to CMS users"[7].

"REDDnet enables researchers to work on large CMS data sets using their own local clusters or OSG resources, even when they lack sufficient storage for the data at the compute site"[8]. The CMS model was designed to facilitate the storage of large amounts of data, but does not provide much freedom for

physics analysis. Time is critically important in the world of computation, and many researchers sharing time at a limited number of storage sites is extremely limiting and inefficient. REDDnet's goal is to de-tether analysis from storage and give physicists the freedom to analyze remotely stored data on their own time. The REDDnet system acts as a conduit between physicists and the CMS infrastructure, freeing them from the bounds and limitations imposed by the data-tethered model. Researchers who have easy access to the data they need, regardless of geography, and can use their own computing resources for individually designed analysis projects will have more freedom to be creative and pursue new physics.

4. GridFTP Testing

I designed load tests for Vanderbilt's gridFTP server. GridFTP is essentially a File Transfer Protocol (FTP) server. The unique function of gridFTP is that it requires users to be a part of the Open Science Grid and authenticate the data transfer with their grid certificate. This provides an important security layer for CMS data storage. Using OSG's Globus toolkit, REDDnet users can manipulate their data within the LoDN system.

a. Purpose

These tests were designed to measure the efficiency and throughput capabilities of gridFTP. I simulated an end-user scenario by uploading files from four of the physics department computers which are connected to the REDDnet gridFTP server via a 1 Gbps connection, which is relatively slow compared to others used in physics research centers. The purpose of these tests was to discover the capabilities and practical limits of our gridFTP server, and deal with any problems or bugs that became apparent. By subjecting the server to a heavy load for sustained periods we can come to a better understanding of the program's reliability, and identify problems that could hinder users from performing high-quality, efficient analyses.

b. Method

The crux of my tests was a gridFTP upload-delete operation wrapped in three layers of code: a run layer, analysis layer, and production layer.

i. **Run Layer** – see Figure 2 for diagram and Appendix 1 for scripts.

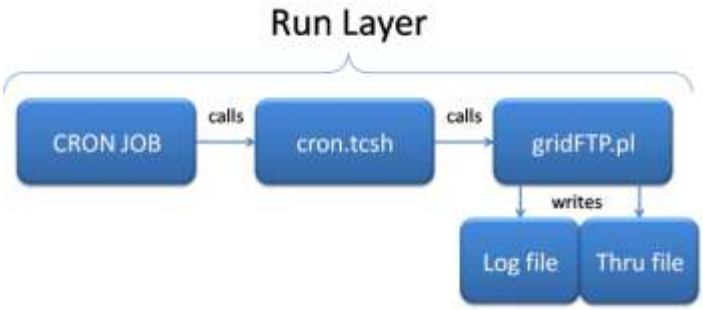


Figure 2: Run Layer Diagram

The bottom layer of scripts included gridFTP.pl and cron.tcsh. In the perl script I defined a loop (lines 50-92) which used the "globus-url-copy" command to upload a file into LoDN with the gridFTP server (line 56). The form of this command is:

```
globus-url-copy file:///tmp/1G gsiftp://se1.accre.vanderbilt.edu/julietest/1G
```

where the source file (labeled by size, i.e. 1G) was located in the /tmp directory; "gsiftp" is the gridFTP server's protocol prefix; the destination was ACCRE's "se1" storage element; and the file was uploaded to the "julietest" directory. Uploaded files were identified with a naming system using dates, times, and sequence number. For example, the eighth 1-GB file in a run that began on March 31st, 2009, at 10:40PM would be uploaded to:

```
gsiftp://se1.accre.vanderbilt.edu/julietest/03312240_1G_8
```

The file upload was timed by setting variables equal to the current epoch time immediately before and after the upload command (lines 55 & 57). The earlier time was subtracted from the later to find the transfer time in seconds (line 58). The file was then deleted from LoDN using the "uberftp" remove command (line 71, shown here for the same example file):

```
uberftp se1.accre.vanderbilt.edu "rm /julietest/03312240_1G_8"
```

Information such as file size, transfer time, error codes, and number of parallel threads were passed after each upload-delete operation to a communal log file which collected all the output for a single day (line 83). Parsing problems arose if multiple operations tried to write output to this file at the same time, essentially clobbering each other and often producing blank lines in the log file, so I separated the log files by size. For example, March 31st had four log files: 0331_1G, 0331_2G, 0331_4G, and 0331_8G.

Each upload-delete operation also wrote a line in a "thru" file (something of an abbreviation for throughput) containing only two entries: file size (50M, 1G, etc) and time of completion (0831, 1324, etc.). These files contained all such information for a single day. (See line 89).

This perl script was called by cron.tcsh, which set up the globus environment (allowing the upload and delete commands to run, see line 6) and collected the date and time information needed in the perl script (lines 8-11). After setting these variables the perl script was called and passed the date (mmdd), time (hhmm), blocksize (file size in bytes divided by 50), and a repeat number used to define how many operations the loop would perform (line 15).

The final participant in the run layer was my local computer systems' cron capabilities. I set up cron jobs on each of four machines (vpac04, vpac05, vpac07, and vpac08) which called cron.tcsh at various minutes of each hour. These times were set up to be spread as equally as possible while accounting for the vast difference in transfer times with file size. Runs with smaller source files were called more times per hour than those testing larger files.

ii. **Analysis Layer –** see Figure 3 for diagram and Appendix 2 for scripts.

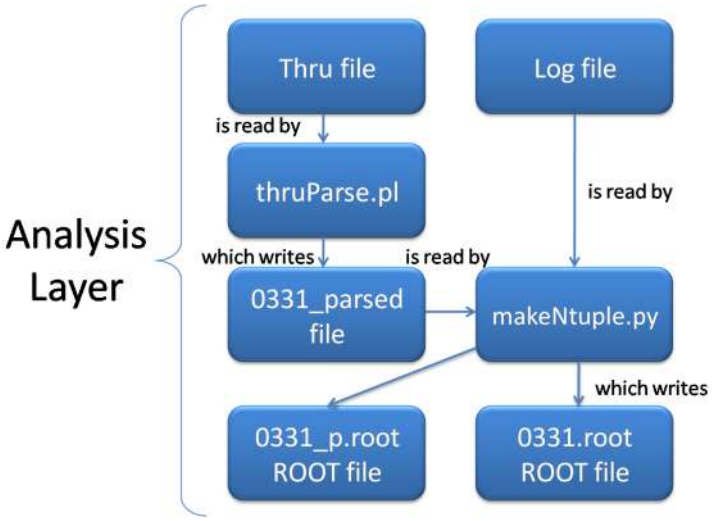


Figure 3: Analysis Layer Diagram (March 31st example)

Every hour the output files from gridFTP.pl were analyzed by the scripts of the analysis layer. The thru file was fed into a perl script called thruParse.pl, which converted character file sizes (i.e. 50M) into bytes (50000000) using regular expressions (lines 24-38, 53-67), and summed over each hour of the day to obtain the total number of bytes transferred each hour (lines 41 & 70). After reading through the log file, the sum of bytes for each hour was divided by 3600 seconds to calculate the throughput in Mbps (lines 43 & 72). The output of this parsing procedure was a file with Hour and Mbps entries, one line for each hour of the day (lines 44 & 73).

The parsed file (labeled date_parsed, i.e. 0331_parsed) and the log file were passed to a python script called makeNtuple.py. The first line of the file was read as the title, and the second line was read as the variable names for the ntuple. For the date_parsed file these variable names were “Hour” and “Mbps” (see sample date_parsed file, line 2). The log files contained the variables “FileSize”, “StartTime”, “EndTime”, “TransferTime”, “Parallel”, and “UploadError” (see sample log file, line 2). All the space-separated values on a line were stored as ntuples corresponding to these variables. The ntuples can be read by ROOT as trees and histograms can be drawn of any variable or combination of variables.

iii. **Graphics Layer** – see Figure 4 for diagram and Appendix 3 for scripts.

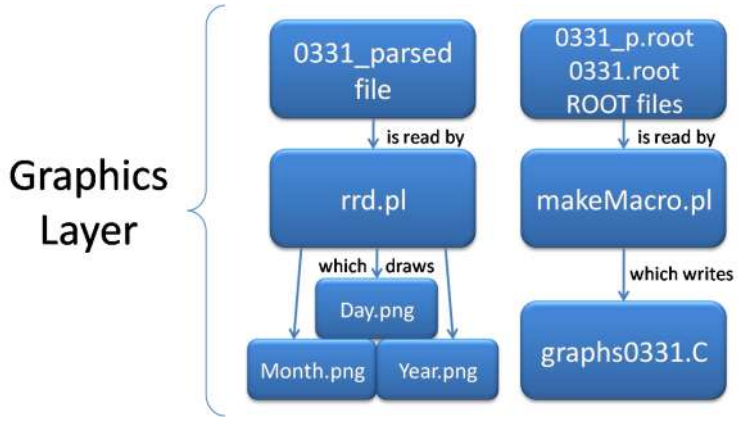


Figure 4: Graphics Layer Diagram (March 31st example)

This layer produced the graphical output for the load tests. All the ROOT files created in the Analysis Layer were read by makeMacro.pl to create a ROOT macro. The date_parsed files created by thruParse.pl were also read by rrd.pl which used the RRD tools to create databases and day, month, and year plots of the total throughput. The coding for the RRD tools is beyond the scope of my project, and I adapted my code from one written by Dr. Engh.

I chose to create the ROOT macros using a perl script to ease the argument-passing process, but other than setting variables makeMacro.pl simply printed a macro which produced a postscript files containing plot images. The date_p.root files were opened and a scatterplot produced of Hour v. Mbps (lines 34-45). For the other plots all the ROOT files for a given day (0331_1G.root, 0331_2G.root, etc) were chained together into one ntuple (lines 47-51). This ntuple contained many variables, but as a default I made histograms of file size, transfer time, and upload error code. Each file size was assigned its own color (lines 53-78), and I superimposed histograms of the transfer times for each file size on the same plot (lines 80-105). I created a second canvas on which I made a histogram of the upload error codes (lines 107-118). A value of 0 represents a successful upload and 1 represents an error.

iv. **Production Layer** – see Figure 5 for diagram and Appendix 4 for scripts.

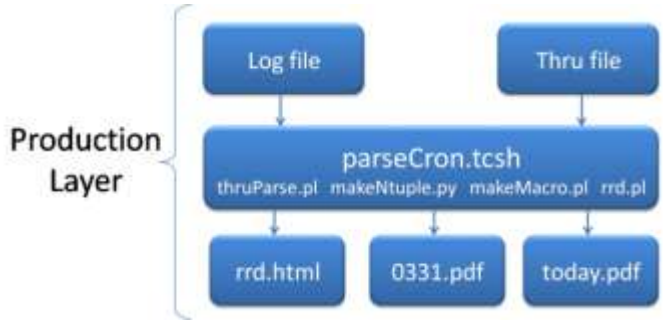


Figure 5: Production Layer Diagram (March 31st example)

This final layer contains the script parseCron.tcsh, which ran the scripts of the Analysis and Graphics layers (lines 8, 12, 20-24, 26). This script also ran a ROOT batch job which executed the macro (line 29). The postscript files were converted to PDF and sent to my ~/web directory. The date.pdf file (with the file size, transfer time, and throughput histograms) was copied to a file called today.pdf which always stores the most current plot (lines 30-38). This layer also produced the RRD graphs mentioned above.

c. Results

The course of these tests was slow and rather tortuous. My main script (gridFTP6.pl) went through six incarnations before reaching its final form. GridFTP5.pl ran the same testing procedure with smaller files (50MB, 100MB, 500MB, and 750MB) with a different delete command. Large files (1G, 2G, 4G, and 8G) and the "uberftp" delete command were added in the final version of the code.

Throughout the testing process I uncovered a seemingly endless chain of errors. The gridFTP server underwent several software upgrades over the course of this project, and REDDnet's depot system was also upgraded considerably. REDDnet now contains many more depots than when I began my tests, and operates a more current version of OSG's Globus toolkit. The uberftp tools were also added near the end of testing. The vast majority of the errors I uncovered related to the functionality of the gridFTP server. Connection to the "se1" storage element caused frequent problems, as did gridFTP's handling of parallel upload streams. The problems have been largely eradicated by the software upgrades made by Dr. Engh and the REDDnet team. We also battled several problems with "hung" jobs, both on my local computers and ACCRE storage element. While temporary fixes have been found, the underlying cause of these stalls is still under investigation.

After a considerable process of optimizing my testing conditions I was able to sustain a throughput between 700 and 840 Mbps for several days, during which time the system produced few errors. Earlier in my testing run I maintained a throughput between 600 and 690 Mbps while transferring smaller files. The throughput increased with the number of files transferred per hour until it reached approximately 800 Mbps. The connection between the local machines and ACCRE is one Gbps, and 800-850 Mbps appears to be the functional limit for file transfer rate over that connection.

Transfer time increased monotonically with file size, although not linearly. It remains unknown why the 8GB files take so much longer to upload than expected by a linear trend. As I pushed the throughput to towards the limit, transfer times tended to increase since my connection to the gridFTP server was essentially clogged. To alleviate stress on the server I added the parallel threads option back into the globus-url-copy command after several days at the high throughput level. This option had caused connection problems early on, but server upgrades fixed this bug. Figure 6 shows the transfer times of each file size over the period from April 4th to April 6th, 2009, when all uploads were serial. Figure 7 adds April 7th to April 9th, when I used 4-thread parallel uploads. Each file size experienced a drop in the mean upload time of between twenty and thirty-three seconds. Decreasing the transfer times with the parallel option also helped decreased the number of simultaneous jobs on the server, since jobs would finish more quickly and overlap less often (Figure 8).

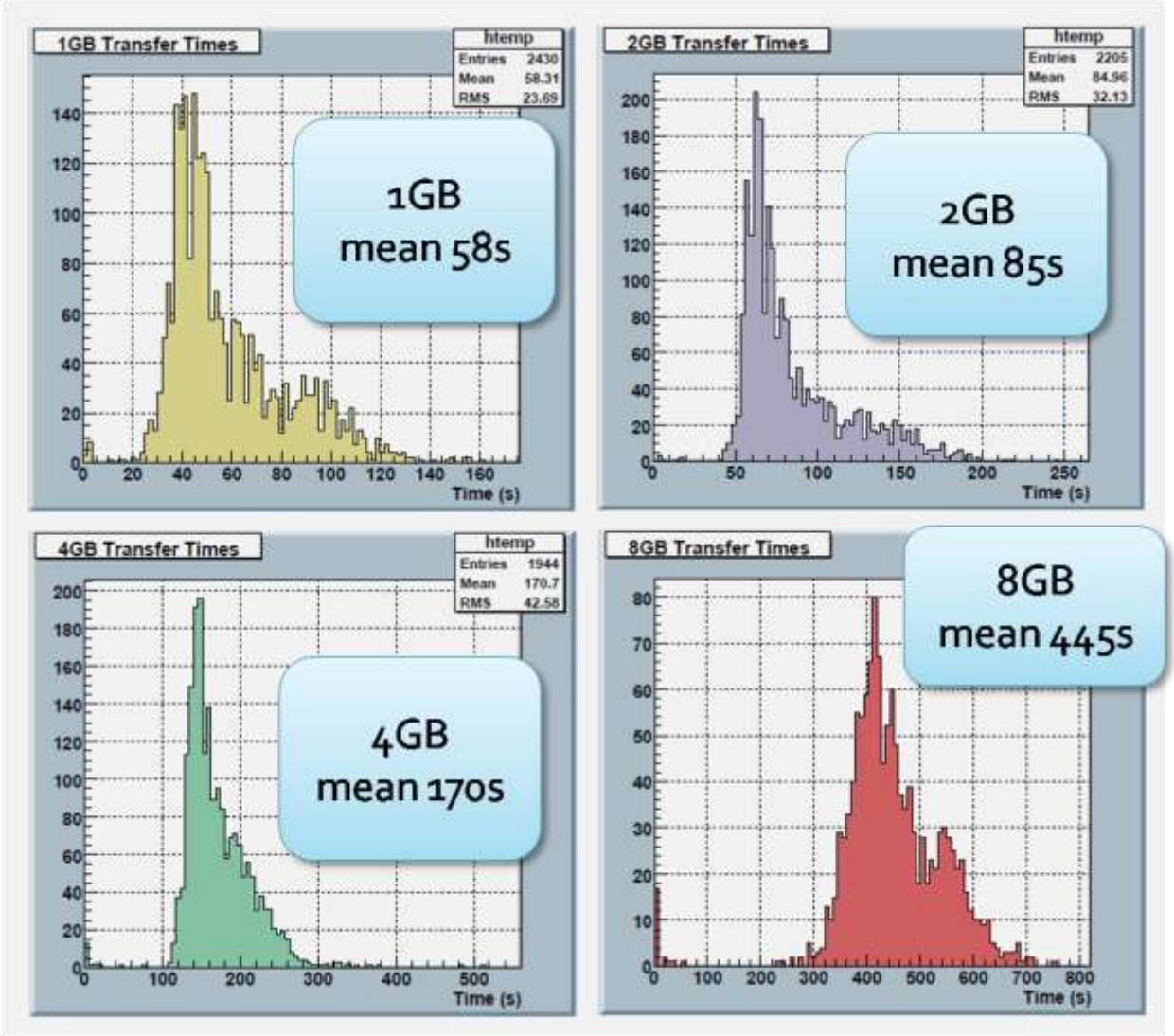


Figure 6: Transfer times by file size for April 4 - 6, 2009. These uploads were done in serial mode.

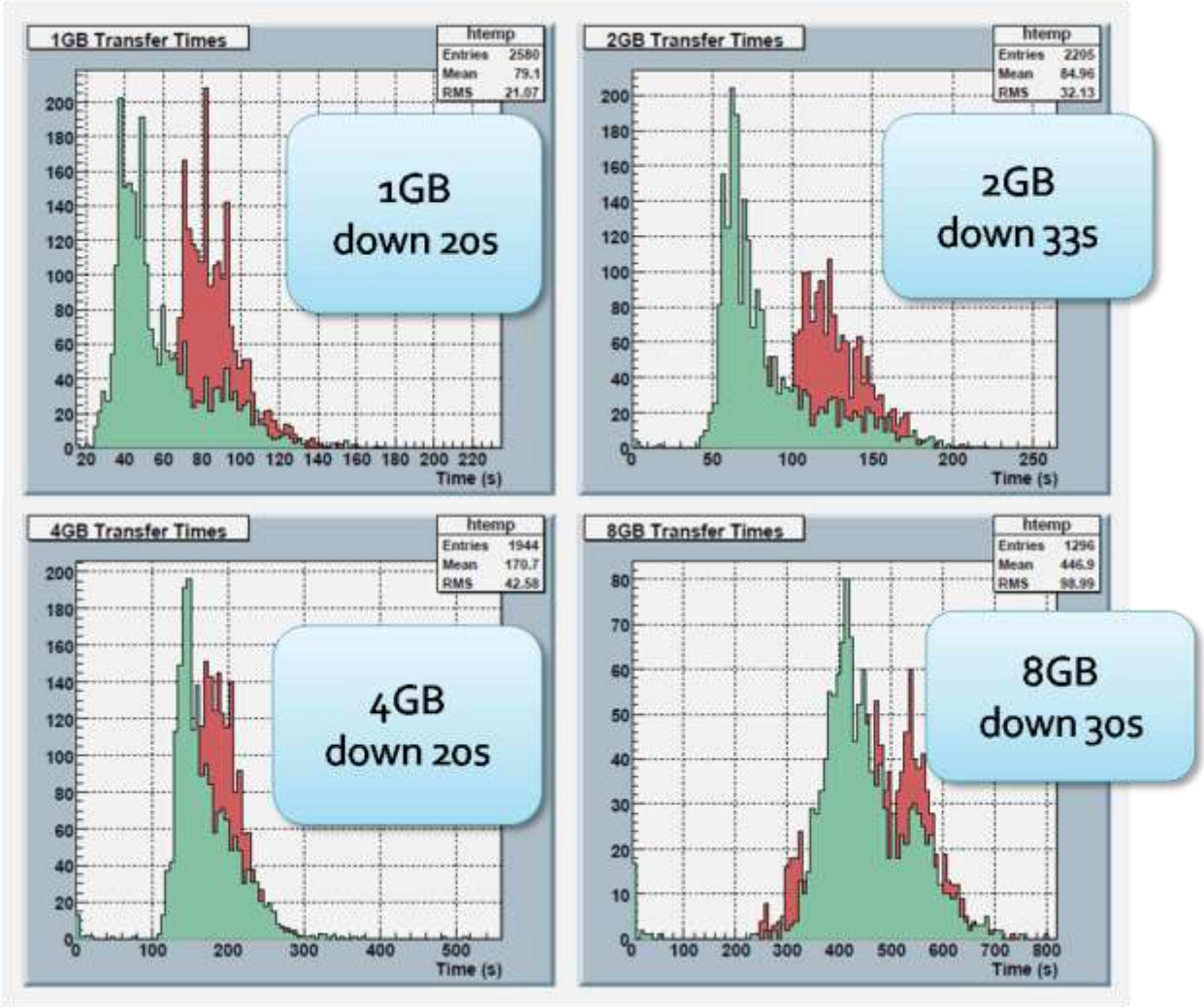


Figure 7: Transfer time by file size for April 4 - 9, 2009. Uploads were changed to parallel mode (4 threads) mid-day April 7. Green indicates parallel uploads, red indicates serial.

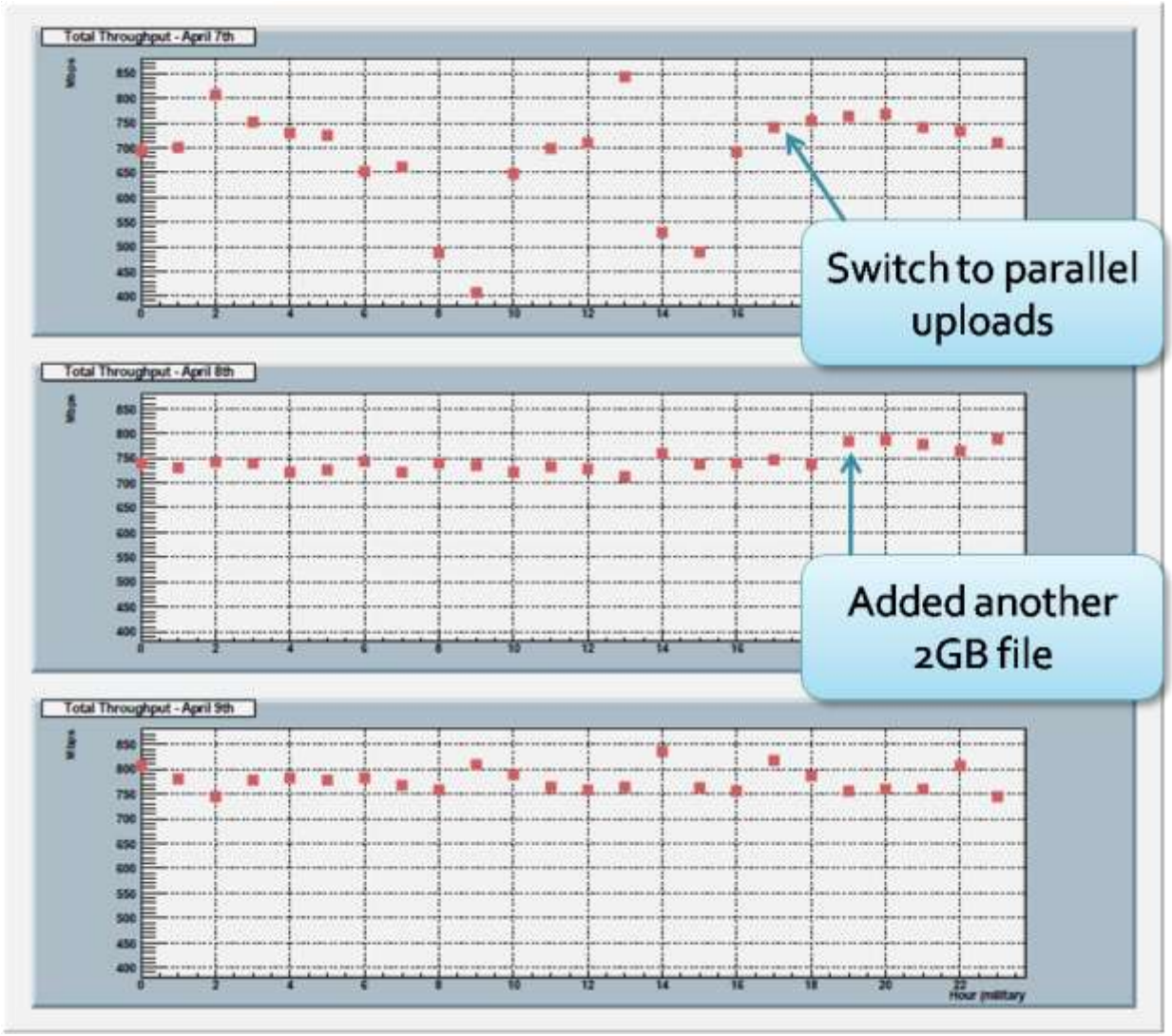


Figure 8: Throughput plots for April 7 - 9, 2009. The change from serial to parallel is marked, as well as an addition of one 2G file to each hour's tests.

Error output was extremely polarized for these tests. Bugs in the gridFTP server, problems with the storage depots, authentication issues, etc, would cause all my tests to fail across the board. Similarly, once these problems were solved (or before they appeared), success rates were virtually 100%. The most notable exception to this observation is the problem with "hung" jobs on the gridFTP server. These jobs do not register as a failed upload because they simply never stop – until Dr. Engh wrote a script to kill transfers lasting longer than ten minutes I saw jobs that had been running for longer than three days. Clearly, this causes significant problems to the user, as hung jobs can pile up on a local machine until its everyday operations are slowed considerably. The cause of this problem is still unknown, and it recurs so inconsistently that it is difficult to pin down. Figure 8 is a composite plot of the error reports for April 4th through 9th. The only errors of interest are those on April 7th, when the load on the gridFTP server caused some connections to be dropped. After assessing these errors I switched to parallel uploads, which solved the problem, as shown above.

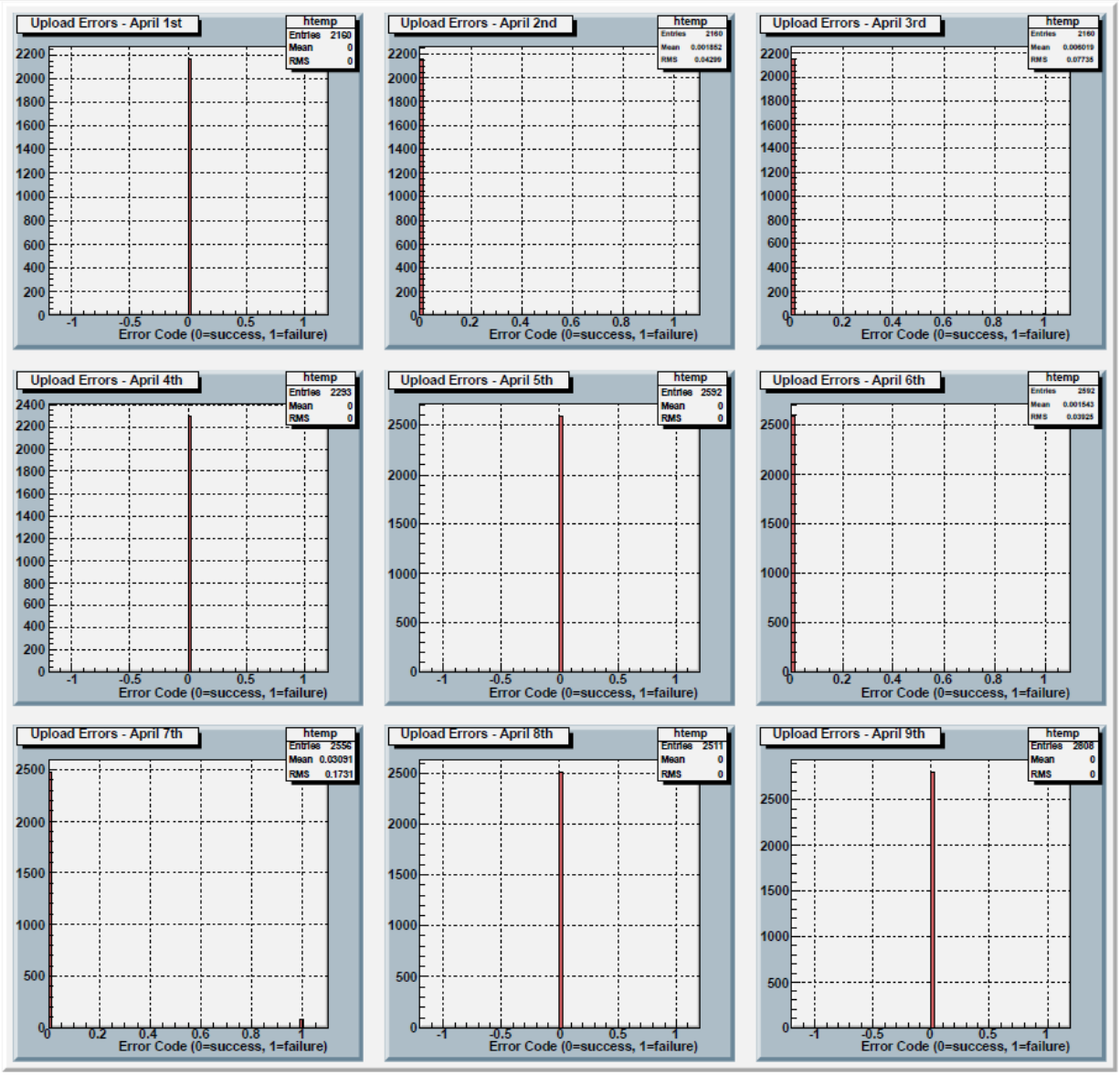


Figure 9: Error code plots for April 1 - 9, 2009. This highlights the fact that failures were extremely rare once the tests were optimized.

d. Project Extensions

Future testing on this track would be useful to continue searching for problems and inefficiencies in the REDDnet storage system that may present themselves to global users, especially the problem of hung jobs. A reverse of my tests would be especially enlightening: uploading source files onto REDDnet depots via LoDN and then running a barrage of tests which download these files to a local system and check their accuracy. In theory, the same transfer rates and throughput should be available in a download scenario, and any discrepancies would be interesting to investigate. Also, I would have liked to test Dr. Engh's LoDN plugin for ROOT in the context of an actual CMSSW analysis job, which would integrate the tools I've tested here with their intended application. I was able to do a detailed study of one step in the process of validating the REDDnet model, and there are many more that would be useful to explore.

5. Conclusions

Vanderbilt's CMS computing system can provide physicists across the globe with a reliable software framework around which to build their analyses. REDDnet depots have been successfully installed across the United States and at CERN, and LoDN provides a powerful way to distribute data safely and securely around the world. My tests hardened one of REDDnet's tools and proved that it can accomplish its goal quickly and efficiently, for sustained periods of time. This tool (the gridFTP server and LoDN) is the primary gateway through which CMS will ingest data into REDDnet. The high, sustained throughputs I achieved are an important measuring stick for a system that also has many qualitative dimensions, such as convenience. I believe that the capabilities demonstrated here, together with developments such as Dr. Engh's ROOT plugin, will meet the needs of REDDnet users. REDDnet is truly a collaborative effort: the data storage system encourages users to collaborate by providing access to files stored across the globe as well as increasing each user's storage power. The network of depots will allow users to keep vast amounts of data actively available, and the lodn:// protocol will open doors to physicists without the local storage power to analyze large datasets. The CRAB program will increase physicist's ability to perform computation-intensive analyses regardless of their location or local resources. Making this system work is a huge process, and I made a specific contribution by testing the gridFTP and LoDN tools. All of our efforts thus far show that the REDDnet model will be successful, and everyone's individual roles are integral to the project as a whole.

The REDDnet system and all that it encompasses is a collaboration of physicists and computer scientists working towards a common goal. Although the CMS detector was designed with many layers of complex triggering systems, the amount of useful data emerging from the machine each day will be staggering. It is of supreme importance that scientists across the globe have resources for storing and handling large amounts of data, and this is what the REDDnet system provides through gridFTP and LoDN. Physics experiments today are extremely large, and scheduling adequate time to use the machinery is no longer feasible for the majority of physicists. REDDnet and its associated programs allow physicists to be creative with the resources available to them, "de-tethering" them from the physical experiment. The ability to remotely analyze CMS data through the global REDDnet collaboration will liberate scientists in new ways to discover exciting new physics from the Compact Muon Solenoid Experiment.

---

[1] CMS Outreach. (2008). *The Grid: the new information superhighway*. Retrieved April 12, 2009, from The Compact Muon Solenoid Experiment: http://cms.web.cern.ch/cms/Detector/Computing/index.html.

[2] CMS Workbook Contributors. (2007). *2.2 CMS Computing Model*. Retrieved April 12, 2009, from The CMS Offline Workbook: https://twiki.cern.ch/twiki/bin/view/CMS/WorkBookComputingModel.

[3] Ibid.

[4] Sheldon, P. (2009). *Working Storage for Data Intensive Collaboration.* unpublished REDDnet Research Highlight.

[5] REDDnet Collaborators. (2008). *REDDnet: Enabling Data Intensive Science in the Wide Area*. Retrieved April 12, 2009, from REDDnet: http://www.reddnet.org/mwiki/index.php/Main_Page

[6] *LoRS: Logistical Runtime System*. (n.d.). Retrieved April 12, 2009, from LoCI: Logistical Computing and Internetworking Lab: http://loci.cs.utk.edu/lors/.

[7] Sheldon, P. (2009). *Working Storage for Data Intensive Collaboration.* unpublished REDDnet Research Highlight.

[8] Ibid.

Works Cited

1. CMS Outreach. (2008). *The Grid: the new information superhighway*. Retrieved April 12, 2009, from The Compact Muon Solenoid Experiment: http://cms.web.cern.ch/cms/Detector/Computing/index.html

2. CMS Workbook Contributors. (2007). *2.2 CMS Computing Model*. Retrieved April 12, 2009, from The CMS Offline Workbook: https://twiki.cern.ch/twiki/bin/view/CMS/WorkBookComputingModel

3. *LoRS: Logistical Runtime System*. (n.d.). Retrieved April 12, 2009, from LoCI: Logistical Computing and Internetworking Lab: http://loci.cs.utk.edu/lors/

4. REDDnet Collaborators. (2008). *REDDnet: Enabling Data Intensive Science in the Wide Area*. Retrieved April 12, 2009, from REDDnet: http://www.reddnet.org/mwiki/index.php/Main_Page

5. Sheldon, P. (2009). *Working Storage for Data Intensive Collaboration.* unpublished REDDnet Research Highlight.

Appendix 1 – Run Layer Scripts and Output Files

I have used emacs notation for wrapped lines: a backslash (\) appears at the end of the line, but is not part of the script, and the second line is indented one space.

1. cron6.tcsh

```
1   #!/bin/tcsh
2
3   # ARGUMENTS: $1 = file_size; $2 = block_size
4
5   source /home/managaje/.cshrc
6   source /usr/local/grid/VDT-Client-1.10.1/setup.csh
7
8   setenv size $1
9   setenv blocksize $2
10  setenv mark `date +"%m%d%H%M"`
11  setenv day `date +"%m%d"`
12
13
14  echo "Starting gridFTP6.pl for ${mark}_${size} with a blocksize of ${b\
      locksize} bytes and 9 runs"
15  perl /home/managaje/gridFTP/PERL/gridFTP6.pl ${mark} ${size} ${blocksi\
      ze} 9 >>& /home/managaje/gridFTP/CRONLOG/cron6/${day}_${size}.log
```

2. gridFTP.pl

For parallel uploads. Before April 7th, 2009, uploads ran in serial mode: line 56 had no –p or –bs options, and the "4" in line 83 was replaced with a "0".

```
1   #!/usr/bin/perl
2
3   use warnings;
4   use strict;
5
6   # ARGUMENTS: 0-datetime; 1-file size; 2-block size; 3-number for loop
7   unless($#ARGV == 3) {
8       die "Not enough arguments: file block_size loop_number";
9   }
10
11  # Set destination variables
12  my $server = "se1.accre.vanderbilt.edu";
13  my $testdir = "julietest";
14
15  # Set globus proxy variables
16  $ENV{"X509_USER_PROXY"} = "/home/managaje/.proxy/x509up_u76168";
17  $ENV{"X509_USER_CERT"} = "/home/managaje/.globus/usercert.pem";
18  $ENV{"X509_USER_KEY"} = "/home/managaje/.globus/userkey.pem";
19
20  # Set LODN environment variables
21  $ENV{"LODN_USER"} = "cms_tier3";
22  $ENV{'PATH'} = '/bin:/usr/bin:/usr/local/grid:/usr/local/grid/reddnet/\
      lodn';
```

```
23
24  # Set up the loop: number of repeats and "failed" test counter
25  my $nrepeat = $ARGV[3] || 60;
26  my $failed = 0;
27
28  # Open log and thru files and print a header if the file is new
29  my $header = 1;
30  my $logdate = `date +"%m%d"`;
31  my $logtime = `date +"%H%M"`;
32  chomp ($logdate);
33  chomp ($logtime);
34  my $thrulog = "/home/managaje/gridFTP/thru/${logdate}.log";
35  my $logfile = "/home/managaje/gridFTP/LOGS/logs6/${logdate}_$ARGV[1].l\
      og";
36
37  if (-e $logfile) {
38      $header = 0;
39  }
40
41  open(my $log, ">>", $logfile) || die("can't open log file: $!");
42  open(my $thru, ">>", $thrulog) || die("can't open log file: $!");
43
44  if ($header != 0) {
45      print $log "$ARGV[0]\nFileSize StartTime EndTime TransferTime Para\
          llel UploadError DownloadError";
46  }
47
48  #====================== START THE LOOP ================================
49
50  for (my $i = 1; $i <= $nrepeat; $i++) {
51
52  print "============================================================\n";
53
54  # Upload the file to LODN and determine success
55  my $time0 = time();
56  my $upload = system("/usr/local/grid/VDT-Client-1.10.1/globus/bin/glob\
      us-url-copy -p 4 -bs $ARGV[2] file:///tmp/$ARGV[1] gsiftp://$server/$\
      testdir/$ARGV[0]_$ARGV[1]_$i");
57  my $time1 = time();
58  my $uptime = $time1-$time0;
59
60  if ($upload == 0) {
61      print "$ARGV[0]_$ARGV[1]_$i uploaded successfully\n";
62      print "$uptime seconds\n";
63  }
64  else {
65      print "upload failed\n";
66      $failed = 1;
67  }
68
69  # Delete the file from LODN
70  my $time2 = time();
71  my $delete = system("/usr/local/grid/VDT-Client-1.10.1/globus/bin/uber\
      ftp $server \"rm /$testdir/$ARGV[0]_$ARGV[1]_$i\"");
72  my $time3 = time();
73  my $downtime = $time3 - $time2;
74
```

```
75  # Calculate analysis variables
76  my $upload2 = $upload/256;          # perl bit correction...
77  my $totaltime = $uptime + $downtime;
78  my $Gbytes = $ARGV[2]*50/1000000000;
79  my $starttime = $time0 - 1230768000;
80  my $endtime = $time3 - 1230768000;
81
82  # Print statements to log and thru files
83  my $brieflog = "\n$Gbytes $starttime $endtime $totaltime 4 $upload2";
84  print $log $brieflog;
85
86  unless($failed) {
87      $logtime = `date +"%H%M"`;
88      chomp ($logtime);
89      print $thru "$ARGV[1] $logtime \n";
90  }
91
92  } #Closes the loop
93
94  close($log);
95  close($thru);
96
97  print "Finished\n";
```

3. Excerpt from thru file (March 31st – 0331.log)

Column 1 = file size; column 2 = time of completion (hhmm). Each line represents one upload-delete operation. See gridFTP.pl, line 89. All operations for one day were recorded in the same thru file.

```
1   2G 0009
2   2G 0011
3   2G 0013
4   2G 0015
5   2G 0017
6   2G 0019
7   8G 0020
8   2G 0021
9   2G 0023
10  2G 0026
11  8G 0028
12  4G 0028
13  4G 0032
14  2G 0033
15  8G 0035
16  2G 0035
17  4G 0036
18  2G 0037
19  2G 0039
20  4G 0039
21  … pattern continues
```

4. Excerpt from log file (March 31st – 0331_1G.log)

Line 1 = title; line 2 = variable names; lines 3+ = variable values for each upload-delete operation. See gridFTP.pl, line 83. Log files were separated by file size to prevent line clobbering.

```
1   title
2   FileSize StartTime EndTime TransferTime Parallel UploadError
3   1 7750803 7750875 72 4 0
4   1 7750875 7750953 78 4 0
5   1 7750953 7751013 60 4 0
6   1 7751013 7751072 59 4 0
7   1 7751072 7751146 74 4 0
8   1 7751146 7751217 71 4 0
9   1 7751217 7751285 68 4 0
10  1 7751285 7751369 84 4 0
11  1 7751369 7751437 68 4 0
12  1 7751701 7751788 87 4 0
13  1 7751788 7751872 84 4 0
14  1 7751872 7751966 94 4 0
15  1 7751966 7752049 83 4 0
16  1 7752049 7752121 72 4 0
17  1 7752121 7752195 74 4 0
18  1 7752195 7752269 74 4 0
19  1 7752269 7752340 71 4 0
20  1 7752340 7752426 86 4 0
21  … pattern continues
```

Appendix 2 – Analysis Layer Scripts and Outputs

1. thruParse.pl

```
1   #!/usr/bin/perl
2
3   use warnings;
4   use strict;
5
6   # This script will parse gridFTP test logs to give overall throughput
7   # for each hour.
8   # Arguments: 0-filename;
9
10  my $hour1 = 1;
11  my $hour2 = 10;
12  my $bytes = 0;
13  my $file = "/home/managaje/gridFTP/thru/$ARGV[0].log";
14
15  open(my $OUT, ">", "/home/managaje/gridFTP/thru/$ARGV[0]_parsed") ||  \
     die("can't open log file: $!");
16  print $OUT "$ARGV[0]\n";
17  print $OUT "Hour Mbps\n";
18
19  for($hour1 = 0; $hour1 < 10; $hour1++) {
20      $bytes = 0;
21      open LOGFILE, $file or die $!;
22      while(<LOGFILE>) {
23        if($_ =~ /(0)($hour1)[0-9][0-9]/) {
24            if($_ =~ /8G/) {
25              $bytes +=8000000000;
26            }
27            elsif($_ =~ /4G/) {
28              $bytes +=4000000000;
29            }
30            elsif($_ =~ /2G/) {
31              $bytes +=2000000000;
32            }
33            elsif($_ =~ /1G/) {
34              $bytes +=1000000000;
35            }
36            else {
37              next;
38            }
39        }
40      }
41      my $bits = $bytes*8;
42      my $Mbits = $bits/1000000;
43      my $Mbits_sec = $Mbits/3600;
44      print $OUT "0$hour1 $Mbits_sec \n";
45      close LOGFILE;
46  }
47
48  for($hour2 = 10; $hour2 < 24; $hour2++) {
49      $bytes = 0;
```

```
50      open LOGFILE, $file or die $!;
51      while(<LOGFILE>) {
52        if($_ =~ /$hour2[0-9][0-9]/) {
53            if($_ =~ /8G/) {
54              $bytes +=8000000000;
55            }
56            elsif($_ =~ /4G/) {
57              $bytes +=4000000000;
58            }
59            elsif($_ =~ /2G/) {
60              $bytes +=2000000000;
61            }
62            elsif($_ =~ /1G/) {
63              $bytes +=1000000000;
64            }
65            else {
66              next;
67            }
68        }
69      }
70      my $bits = $bytes*8;
71      my $Mbits = $bits/1000000;
72      my $Mbits_sec = $Mbits/3600;
73      print $OUT "$hour2 $Mbits_sec \n";
74      close LOGFILE;
75  }
76
77  close $OUT;
```

2. Sample date_parsed file (0331_parsed) - Output from thruParse.pl, lines 16-17, 44 and 73.

```
1   0331
2   Hour Mbps
3   01 426.666666666667
4   02 435.555555555556
5   03 431.111111111111
6   04 435.555555555556
7   05 435.555555555556
8   06 435.555555555556
9   07 435.555555555556
10  08 435.555555555556
11  09 426.666666666667
12  10 440
13  11 422.222222222222
14  12 497.777777777778
15  13 508.888888888889
16  14 526.666666666667
17  15 513.333333333333
18  16 500
19  17 502.222222222222
20  18 511.111111111111
21  19 513.333333333333
22  20 504.444444444444
23  21 517.777777777778
24  22 524.444444444444
25  23 511.111111111111
```

3. makeNtuple.py

```
"""
   Build ROOT Ntuple from other source.
   This program reads the `aptuple.txt' file row by row, then creates
   the Ntuple by adding row by row.
"""

import sys, string
from ROOT import TFile, TNtuple

ifn = sys.argv[1]
ofn = sys.argv[2]

print 'opening file', ifn, '...'
infile = open( ifn, 'r' )
lines  = infile.readlines()
title  = lines[0]
labels = string.split( lines[1] )

print 'writing file', ofn, '...'
outfile = TFile( ofn, 'RECREATE', 'ROOT file with an NTuple' )
ntuple  = TNtuple( 'ntuple', title, string.join( labels, ':') )

for line in lines[2:]:
    words = string.split( line )
    row = map( float, words )
    apply( ntuple.Fill, row )

outfile.Write()

print 'done'
```

Appendix 3 – Graphics Layer scripts and outputs

1. rrd.pl

```
#!/usr/bin/perl

# this script gets tcp input and output bytes for rddtool
#          -dje feb 09
#
# 1. You need to implement a method to find 2 values: bytes in and
# out.  2. Use rrd like defined here, no need to change.  3. Set up a
# cronjob to run this script every 5 mins (or match with --step option
# below) 4. Set up an auto-refresh web page to display the charts.

use strict;

# set your rrd related locations
my $rrdtool="/home/managaje/RRD/bin/rrdtool"; #the exe
my $rrdfile="/home/managaje/RRD/gFTPthru.rrd"; #the db
my $webdir="/home/managaje/web/rrd"; #plots go here

my $sys_cmd;
my $file = "/home/managaje/gridFTP/thru/$ARGV[0]_parsed";
my $currentThru;

open INPUT, $file or die $!;
while (<INPUT>) {
    if ($_ =~ /^$ARGV[1] /) {
      if ($_ =~ / ([0-9]+\.?[0-9]*)/) {
          $currentThru = $1;
      }
    }
}
print "$ARGV[1]\n";        #for checking the input...
print "$currentThru\n";

# execute the rrdtool commands (create, update, graph)

# create the db if it doesn't exist yet
if (! -e "$rrdfile") {
# create db with 2 vars, with 3 avgs saved: every data pt (1), last 6,
# last 144
#    print "creating new db\n"; #debug
    $sys_cmd="$rrdtool create $rrdfile --step 3600"
      ." DS:thru:GAUGE:3900:0:0.18446744073e+19"
      ." RRA:AVERAGE:0.5:1:24"
      ." RRA:AVERAGE:0.5:24:31"
      ." RRA:AVERAGE:0.5:24:365";
#    print "system command = $sys_cmd\n"; #debug
    system($sys_cmd);
}

$sys_cmd="$rrdtool update $rrdfile N:${currentThru}";
system($sys_cmd);

```

```
52  my $graph_def = "DEF:inv=$rrdfile:thru:AVERAGE "
53      ."AREA:inv#00FF00:throughput_Mbps ";
54
55  my $graph = 0; # dummy to hold graph command stdout
56  # make 3 plots: day, month, year
57  $graph = `$rrdtool graph $webdir/gFTP_day.png  --title "GridFTP Transf\
      ers 24 hour activity" --start -24h $graph_def`;
58  $graph = `$rrdtool graph $webdir/gFTP_month.png --title "GridFTP Trans\
      fers 30 day activity" --start -30d $graph_def`;
59  $graph = `$rrdtool graph $webdir/gFTP_year.png  --title "GridFTP Trans\
      fers 1 year activity" --start -1y $graph_def`;
60
61  exit;
```

2. makeMacro.pl

```
1   #!/usr/bin/perl
2
3   # Creates a ROOT macro for ARGV[0].root and produces user-defined plots
4   # Arguments: 0-root file prefix ; 1-plot 1 label ; 2-plot 2 label ;
5   # 3-plot 3 label; etc... Plots 3 and 4 can each take two labels.
6
7   unless($#ARGV >= 0) {
8     die "Not enough arguments: root file";
9   }
10
11  my $plot1 = $ARGV[1] || "FileSize";
12  my $plot2 = $ARGV[2] || "TransferTime";
13  my $plot3 = $ARGV[3] || "UploadError";
14
15  my $macrofile = "/home/managaje/gridFTP/root/macros/graphs$ARGV[0].C";
16  open(my $macro, ">", $macrofile) || die("can't open macro file: $!");
17
18  # EVERY LINE FROM HERE ON BEGINS print $macro " AND ENDS WITH  \n";
19  # FOR CLARITY THESE WILL BE REMOVED AFTER LINE 24 – I HAVE ALSO REMOVED
20  # ESCAPE \'s SO YOU CAN SEE THE TRUE SYNTAX OF THE ROOT MACRO.
21
22  print $macro "void graphs$ARGV[0](){\n";
23  print $macro "\n";
24
25  TCanvas *c1 = new TCanvas("c1", "c1",10,10,2000,1800);
26
27  pad1 = new TPad("pad1","$plot1",0.05,0.52,0.48,0.95);
28  pad2 = new TPad("pad2","$plot2",0.52,0.52,0.95,0.95);
29  pad3 = new TPad("pad3","Throughput",0.05,0.05,0.95,0.48);
30  pad1->Draw();
31  pad2->Draw();
32  pad3->Draw();
33
34  pad3->cd();
35  pad3->SetGrid();
36  pad3->SetFillColor(33);
37  pad3->SetFrameFillColor(19);
38  TFile f("root_p/$ARGV[0]_p.root");
39  ntuple->SetMarkerStyle(21);
40  ntuple->SetMarkerColor(46);
41  ntuple->Draw("Mbps:Hour");
42  htemp->SetTitle(\"Total Throughput\");
```

```
43  htemp->GetXaxis()->SetTitle(\"Hour (military)\");
44  htemp->GetYaxis()->Set(100,380,880);
45  c1->Update();
46
47  TChain chain("ntuple");
48  chain.Add("root/$ARGV[0]_1G.root");
49  chain.Add("root/$ARGV[0]_2G.root");
50  chain.Add("root/$ARGV[0]_4G.root");
51  chain.Add("root/$ARGV[0]_8G.root");
52
53  pad1->cd();
54  pad1->SetGrid();
55  pad1->SetFillColor(33);
56  pad1->SetFrameFillColor(19);
57  gStyle->SetOptStat(0);
58  chain.SetFillStyle(0);
59  chain.SetLineColor(19);
60  chain.Draw("$plot1");
61  htemp->SetTitle(\"File Size Distribution\");
62  htemp->GetXaxis()->SetTitle(\"Size (GB)\");
63  pad1->Update();
64  chain.SetFillStyle(1001);
65  chain.SetFillColor(46);
66  chain.SetLineColor(kBlack);
67  chain.Draw("$plot1","FileSize == 8","same");
68  pad1->Update();
69  chain.SetFillColor(30);
70  chain.Draw("$plot1","FileSize == 4","same");
71  pad1->Update();
72  chain.SetFillColor(40);
73  chain.Draw("$plot1","FileSize == 2","same");
74  pad1->Update();
75  chain.SetFillColor(41);
76  chain.Draw("$plot1","FileSize == 1","same");
77  pad1->Update();
78  c1->Update();
79
80  pad2->cd();
81  pad2->SetGrid();
82  pad2->SetFillColor(33);
83  pad2->SetFrameFillColor(19);
84  gStyle->SetOptStat(0);
85  chain.SetFillStyle(0);
86  chain.SetLineColor(19);
87  chain.Draw("$plot2","","",10000);
88  htemp->SetTitle(\"Transfer Times\");
89  htemp->GetXaxis()->SetTitle(\"Time (s)\");
90  pad2->Update();
91  chain.SetFillStyle(1001);
92  chain.SetFillColor(46);
93  chain.SetLineColor(kBlack);
94  chain.Draw("$plot2","FileSize == 8","same");
95  pad2->Update();
96  chain.SetFillColor(30);
97  chain.Draw("$plot2","FileSize == 4","same");
98  pad2->Update();
99  chain.SetFillColor(40);
```

```
100  chain.Draw("$plot2","FileSize == 2","same");
101  pad2->Update();
102  chain.SetFillColor(41);
103  chain.Draw("$plot2","FileSize == 1","same");
104  pad2->Update();
105  c1->Update();
106
107  TCanvas *c2 = new TCanvas("c2", "c2",10,10,2000,1800);
108
109  pad4 = new TPad("pad4","$plot3",0.05,0.05,0.95,0.95);
110  pad4->Draw();
111  pad4->cd();
112  pad3->SetGrid();
113  pad4->SetFillColor(33);
114  pad4->SetFrameFillColor(19);
115  chain.SetLineColor(kBlack);
116  chain.SetFillColor(46);
117  chain.Draw("$plot3");
118  c2->Update();
119
120  c1->Print("$ARGV[0].ps", "psLandscape");
121  c2->Print("$ARGV[0]error.ps");
122
123  # RESUME REGULAR PERL SYNTAX
124  close($macro);
```

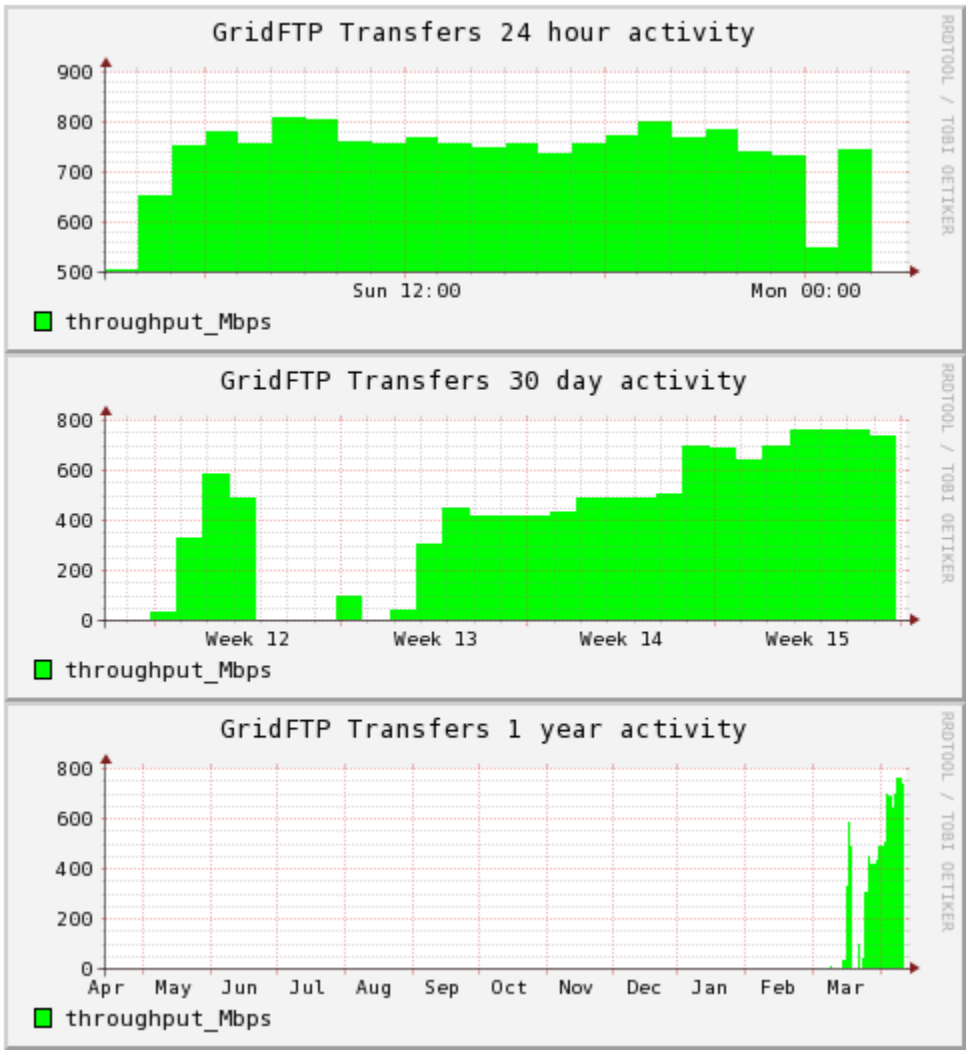
Appendix 4 – Production Layer scripts and images

1. parseCron.tcsh

```
1    #!/bin/tcsh
2
3    # Parse the throughput data
4    cd /home/managaje/gridFTP/PARSE
5    setenv day `date +"%m%d"`
6    setenv hour `date +"%H"`
7
8    thruParse2.pl $day
9
10   # Update the RRDtool database and graphs
11   cd /home/managaje/gridFTP/PERL
12   perl rrd.pl $day $hour
13
14   # Set ROOT environment, create ROOT files and macros
15   cd /home/managaje/cms/CMSSW_2_0_6/src
16   source /home/cmscode/setup.csh
17   cmsenv
18
19   cd /home/managaje/gridFTP/root
20   python makeNtuple.py /home/managaje/gridFTP/thru/${day}_parsed /home/m\
       anagaje/gridFTP/root/root_p/${day}_p.root
21   python makeNtuple.py /home/managaje/gridFTP/LOGS/logs6/${day}_1G.log /\
       home/managaje/gridFTP/root/root/${day}_1G.root
22   python makeNtuple.py /home/managaje/gridFTP/LOGS/logs6/${day}_2G.log /\
       home/managaje/gridFTP/root/root/${day}_2G.root
23   python makeNtuple.py /home/managaje/gridFTP/LOGS/logs6/${day}_4G.log /\
       home/managaje/gridFTP/root/root/${day}_4G.root
24   python makeNtuple.py /home/managaje/gridFTP/LOGS/logs6/${day}_8G.log /\
       home/managaje/gridFTP/root/root/${day}_8G.root
25
26   makeMacro3.pl $day $1 $2 $3 $4 $5 $6
27
28   # Execute the macro and put plots on the web
29   root -l -b -q macros/graphs${day}.C
30   ps2pdf ${day}.ps
31   ps2pdf ${day}error.ps
32   mv ${day}.ps ps
33   mv ${day}error.ps ps
34   mv ${day}.pdf pdf
35   mv ${day}error.pdf pdf
36   cp pdf/${day}.pdf /home/managaje/web/
37   cp pdf/${day}error.pdf /home/managaje/web
38   cp /home/managaje/web/${day}.pdf /home/managaje/web/today.pdf
```

2. rrd.html screen-shot

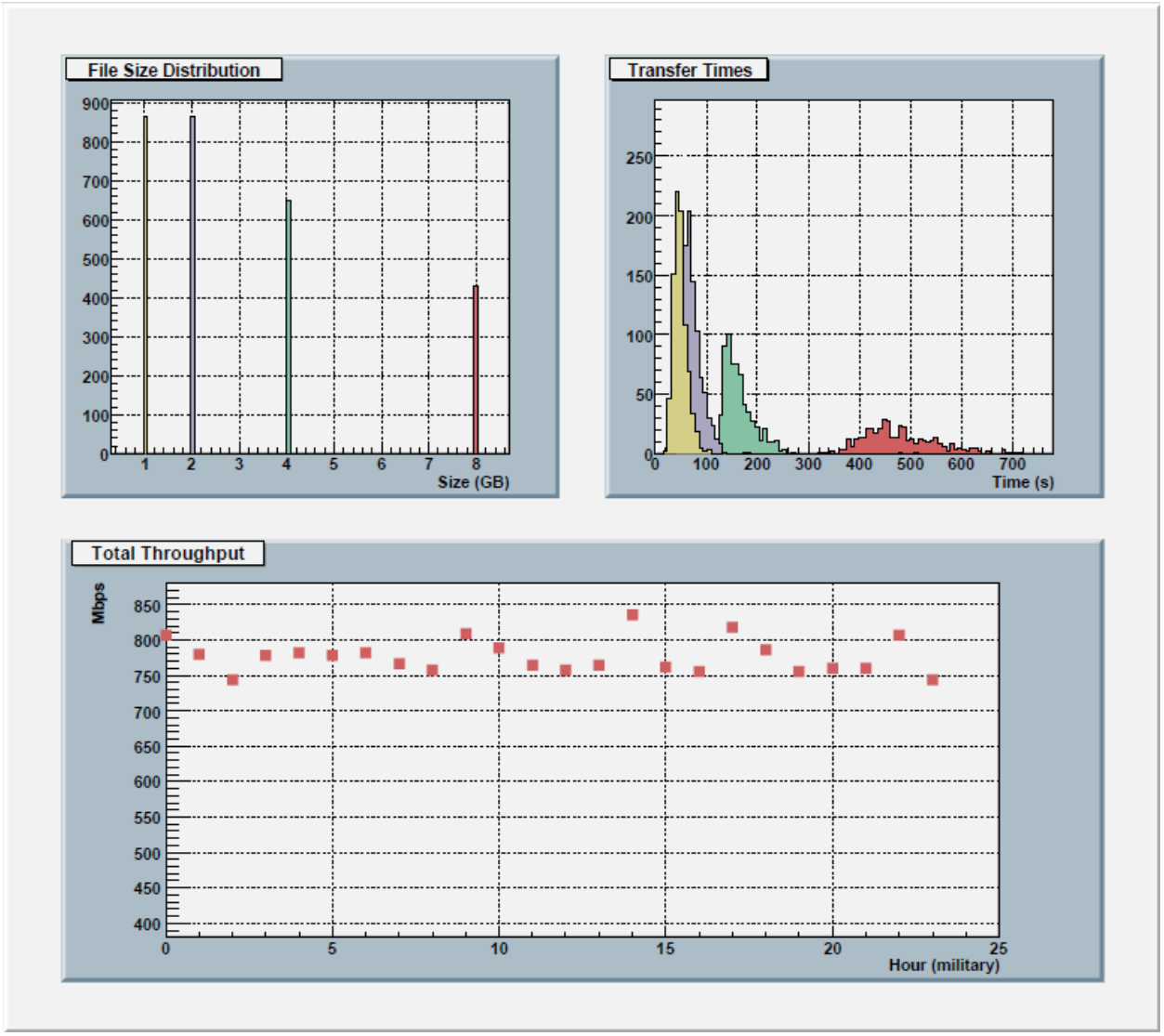
This shot taken at 3:16 AM, April 13, 2009. It is not clear to me what causes the drop in the graph at midnight, which occurred every day. I am relatively unfamiliar with RRD tools, and used this plot to check general status. The ROOT image provides a more precise indication of the throughput.



This image is updated every hour - the most current values can be found by visiting:

http://www.hep.vanderbilt.edu/~managaje/rrd.html

3. today.pdf (0409.pdf) image



This image is updated every hour – the most current values can be found by visiting:

http://www.hep.vanderbilt.edu/~managaje/today.pdf

An archive of each day's output is saved as mmdd.pdf, i.e. 0331.pdf, in the same directory.