

# Experience with Bursty Workflow-driven Workloads in LEAD Science Gateway

Yiming Sun, Suresh Marru, Beth Plale  
*Computer Science Department, Indiana University*  
{yimsun, smarru, plale}@cs.indiana.edu

---

## ABSTRACT

Workloads created from typical meteorology forecast workflows in LEAD are bursty in nature. As the number of simultaneous workflows increases, the burstiness aggravates and we start to encounter a problem in file transfer failure that can cause the file transfer servers to crash. Our solution addresses the problem by smoothing bursts in file transfer workloads and control for limited resources at the file transfer servers. With this new solution in place, we achieved 0% failure rate whereas it used to be as high as 20%.

## 1 INTRODUCTION

When running workflows through a Science Gateway that utilizes remotely located supercomputers, user demands for interactivity mandate that data products move off the remote resource and into the user's personal workspace as soon as they are generated. The computation resources and data are often located at supercomputer centers which are geographically separated from where the Gateway stores data on behalf of users. The users typically interact with the system services and resources via web interfaces, and through these, access and manage their personal workspaces [10, 6]. While various tools have been developed to assist users in monitoring the status of workflow executions [8], as usage and experience has shown, users prefer to focus on the emergence of data products in their workspace as an indication of workflow completion because these are the end results that enable them to move on to next steps.

Meteorologists using the LEAD Science Gateway [13] typically launch weather forecast workflows that cover a period of 24 – 36 hours. The workflows deploy parallel forecast models as application services that generate data each hour over the forecast time duration, with each generated data file being up to 1.5GB in size. In addition, the forecast workflows also generate about 10 - 20 files of smaller sizes throughout execution. Researchers and graduate students may also launch data mining workflows that analyze real time radar data for storm signatures. These latter workflows are shorter running, and generate a number of smaller files. A mix of these two types of workflows characterizes typical Gateway usage at any point in time. This typical usage, or workload, for a 1-hour period is

shown in Fig. 1. The bursty nature is evident. However, as the number of simultaneous workflows increases, we encounter a problem in file transfer failure that can, among other behaviors, cause the file transfer servers to crash.

In this paper, we discuss our experience with file transfer performance in the LEAD Science Gateway as it experienced dramatic growth in a number of simultaneous workloads. We discuss and demonstrate the problems encountered, and offer a solution. We demonstrate the effectiveness of the solution under a realistic workload.

The remainder of this paper is organized as follows: in Section 2 we discuss research related to file transfer. In Section 3 we briefly introduce our file transfer service. Section 4 is a detailed discussion on the scalability problem caused by the workload as well as analysis of file transfer behavior data captured from system logs. In Section 5 we discuss our solution to the problem based on our analysis. In Section 6 we present experimental results after the changes are made. Finally in Section 7, we draw our conclusions and briefly discuss some possible future improvements.

## 2 RELATED RESEARCH

In distributed systems where resources are distributed across geographic distances, data movement is a necessity and consequently, there is considerable research work around this topic.

GridFTP [1] is a file transfer mechanism for moving files in a grid. It is a popular component in Globus Toolkit, and many other file movement tools support it. However, since it is an implementation of a protocol, and not a service, components that wish to perform file transfer must be tightly coupled with a GridFTP client or some library containing it. Should the system choose a different transfer mechanism in the future, all these components must be modified.

Trebuchet [12], developed at NCSA, is a file mover tool that supports Unix-style file operations, and a rich set of transfer protocols. Although its APIs are not specific to any particular transfer protocols, Trebuchet is built as a set of libraries that must be included by the components using it.

We have examined in some detail the Lustre file system [4, 7]. Unlike GridFTP, Lustre imposes almost no code change to cyberinfrastructure components because it is mounted transparently as local file system. In addition, a recent study [9] shows very promising performance by Lustre on the TeraGrid. However, after careful evaluation, we determined that although service components would use Lustre transparently as a local file system, there would be heavy dependencies at lower levels. A repository cannot be accessed unless it is mounted through Lustre, and sometimes this may not be achievable for nodes outside the TeraGrid. It thus imposes a restriction on where files can reside.

Kangaroo is a data-movement system with very simple APIs that allows multiple Kangaroo servers to be chained to maximize performance and availability while trading off file consistency [11]. While Kangaroo appears to be best suited to applications that perform significant amounts of read and write file I/O, the LEAD Science Gateway does not require such functionalities because all files are immutable and are only transferred in their entirety.

The Application Hosting Environment (AHE), a part of the OMII stack, contains a file staging service for moving data around. It assumes light-weight clients that do not have Globus installed, and supports HTTP, HTTPS, SOAP, and GridFTP protocols [3].

The Comprehensive File Transfer (CFT) service is a component in the GridPort Toolkit project [2]. This service uses GridFTP as its underlying transfer mechanism and allows file transfer requests to be submitted in batch. However, CFT has a dependency on the GridPort Repository for user's proxies and requires its client to be on the same file system as the repository, something which we see as a limitation.

Reliable File Transfer (RFT) [5] is a part of Globus Toolkit distribution. Built on top of GridFTP, RFT offers reliability in file transfer by storing transfer states and retrying upon failures, a feature not supported in many other tools. Since the Globus Toolkit is part of the LEAD Science Gateway stack, RFT is a viable piece of our solution to the file transfer failures we are seeing.

### 3 FILE TRANSFER SUPPORT IN GATEWAY

In LEAD Science Gateway, data products generated by workflows are transferred from scratch space on a supercomputer into a personal workspaces using our data movement and naming service. This service serves two primary purposes: to carry out file transfers and to perform logical to physical name resolutions. As a light-weight service with simple APIs to abstract the underlying transfer mechanism, this service spares its clients from having to install a heavy dependency stack (e.g. Globus Toolkit) in order to transfer files, or

to write code that is specific to some transfer mechanisms. The name resolution function allows it to separate logical filenames from their physical locations, so in the event of storage migration, archived files will still be accessible with minimal update efforts.

Data products are hosted across various sites, and they are diverse in many aspects, including security settings, transfer protocols supported, and configurations of hosting software used. In order to have access to all these sites, the data movement and naming service is designed with great flexibility to allow different host specific policies as well as underlying transfer mechanisms to be used through simple configurations. Currently, we have it configured to use RFT as the dominant transfer mechanism unless a site-specific policy overrides it.

### 4 IMPLICATIONS OF INCREASED WORKLOAD

We have experienced frequent file transfer failures in our system, when the load is high, with the failure rate as high as 20%. The error messages accompanying the transfer failures are misleading, stating that the GridFTP servers refuse authentication requests, leading us to conclude the problems are security related. However, knowing that many other transfers between the same pair of nodes finished successfully, we immediately eliminate security as the root cause and suspect that the servers were overloaded beyond their capacities.

Our first attempt in fixing the problem was to use a load-balancing solution. Most data products from the workflows are generated and stored in the scratch space on BigRed at Indiana University, and the users' personal data repository is also accessed via BigRed. At that time, all transfers were passed through a GridFTP front node that fronted four additional, addressable GridFTP server nodes. Concluding the front node was overwhelmed, we experimented with accessing all five servers in a round-robin fashion. This solution was, in hindsight, somewhat naïve, and was quickly retracted. One feature in our data movement and naming service is the concept of logical stores, where a symbolic name can be used for certain data store (e.g. personal data repository). Clients that wish to transfer files to a logical store simply provide this symbolic name as the destination, and it is up to the data movement service to figure out the physical URLs of the store. With this feature, we were able to easily add round-robin capability to destination URL selection module in the data movement service and also configured it to directly address all GridFTP nodes to access the personal data repository.

Unfortunately the services generating these data products have no knowledge of logical stores in our data movement service, and could only use physical

URLs as the source locations of these data products, so it was not possible to perform round-robin selections on the source URLs, and GridFTP servers were still crashing.

With a more careful review of the problem, we realized that while trying to deliver data products into users' personal workspaces as promptly as possible, we were inadvertently treating the resources on TeraGrid as unlimited. A solution lay in designing a strategy that would optimally utilize the resources while staying within their limits, where "limits" is known to be a constantly changing barrier.

To better understand what was happening in the system when the transfer failures occurred, we obtained system logs and extracted file transfer activity data for analysis. All graphs presented in this section are generated from transfer activity data in log files *before* any design changes were made to the system.

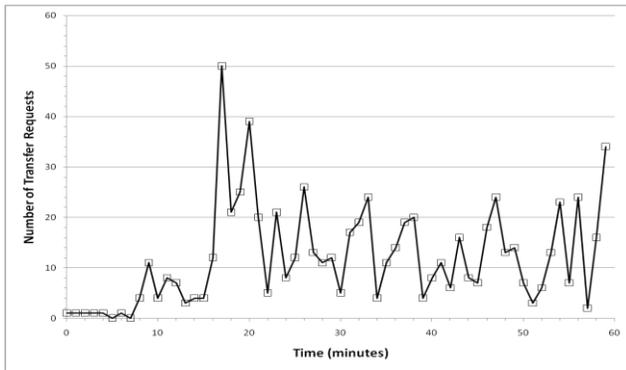


Fig.1. Baseline workload: file transfer requests generated by 55 simultaneous workflows.

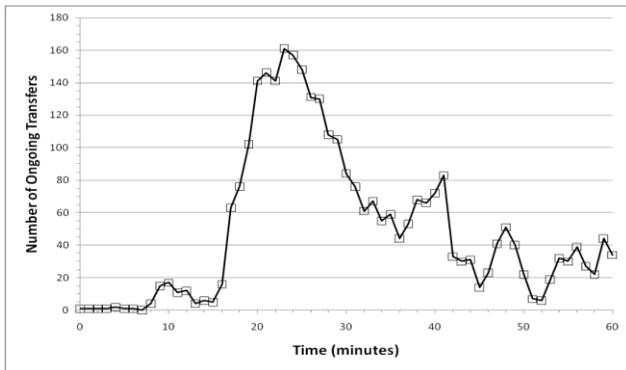


Fig.2. Concurrent file transfer activity measured at specific moments in time.

Fig. 1 plots the number of file transfer requests entering the system over an hour. These data were extracted from system logs and reflect real workflow activities. During this particular hour, a total of 55 workflows were launched, 22 of which were data mining workflows for storm detection and the rest were WRF forecast workflows. The total number of files being transferred during this hour was 723. As can be seen, transfer requests form a bursty pattern,

characterized by periods of little activity followed by spikes reaching as high as 40 or 50. This means that while the average number of transfer requests over the observed period may not be high, the short bursts of incoming requests indicated by the peaks in the graph could drive the GridFTP servers beyond their capacities. On the other hand, during the time intervals indicated by the valleys in the graph, these servers may be underutilized.

Furthermore, due to the files sizes, file transfer can be a time consuming process. After a request is made, the actual workload for transferring files could be ongoing for the next several minutes, during which more requests are being made. Even with no incoming requests, the load on the system may still be high due to earlier requests.

Fig. 2 shows the number of ongoing transfers over time by taking into consideration of the time span of transfers. In this plot, the number of concurrent transfers is calculated for every minute within the hour by adding the total number of transfers from the previous minute and the number of new requests from the current minute, and then deducting the number of transfers ended during the previous minute. Comparing this graph with Fig. 1, we can see that during the period between 16 minutes and 30 minutes into the hour, while the number of requests fluctuated between 5 and 50, the number of concurrent transfers stayed mostly above 100 and reached as high as 160. Since the network bandwidth is a limited resource, the time it takes to finish a transfer is directly affected by the number of concurrent transfers. In other words, when there are bursts in the number of transfer requests, the servers may be saturated for a longer period, until the transfers either finish successfully or fail due to overwhelmed servers.

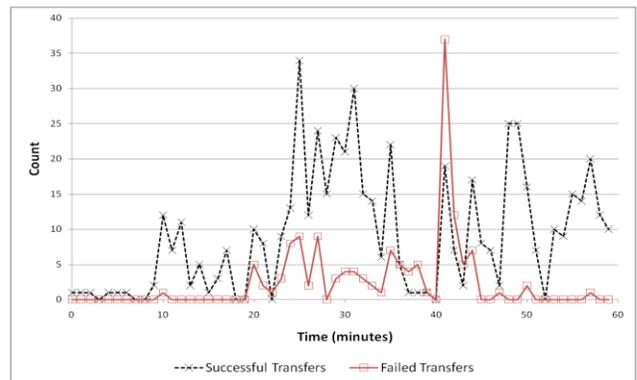


Fig.3. Transfer successes and failures occurring during every minute within an hour.

Fig. 3 plots the number of file transfers finishing successfully and those failing during every minute within the workload hour. Note that the line representing successful transfers roughly resembles the shape in Fig. 1, but with a small right-shift. This is

because the number of successful transfers corresponds roughly to the number of requests, and the time span of the transfer process shifts them right on the time axis.

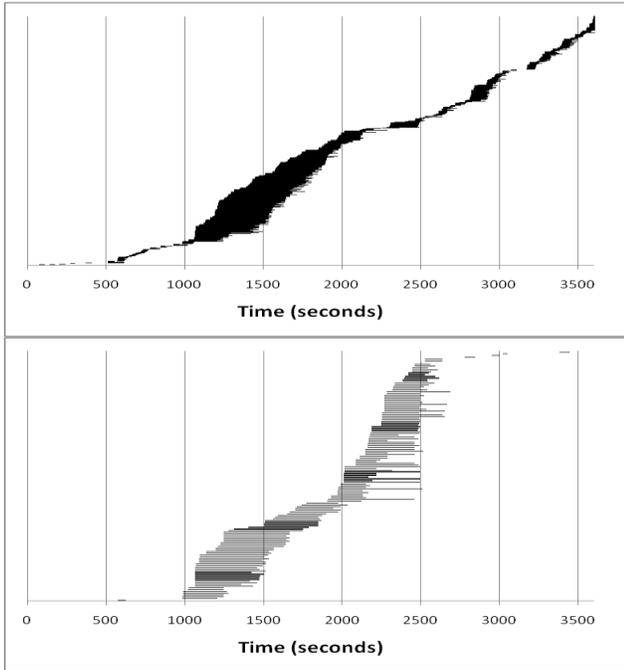


Fig. 4. Time span of all file transfers. The top graph shows successful transfers and the bottom one shows failed transfers. Each horizontal line represents one file transfer from starting second to ending second.

Also note, however, that as the number of requests increases, so does the number of transfer failures. At first it may seem unusual that the highest number of failures occurred approximately 41 minutes into the hour, where the number of concurrent transfers is not very high during that interval. This is because RFT has built-in retry capabilities, and when a transfer attempt fails, it will retry a specified number of times before giving up and declaring the transfer failed. Between the 18 and 40 minute marks, there is a heavy load on the system and many of the transfer attempts and retries fail during this period. When the maximum number of retries is reached without success, those transfers are declared failed although the system has quieted down at the time. Fig. 4 displays in Gantt chart form the exact starting and ending time of these transfers. The horizontal axis represents the number of seconds elapsed in the hour, and the vertical axis represents all file transfers. Because there are 723 transfers, the ones that finished successfully and those that failed are plotted on separate graphs for clarity.

## 5 SOLUTION

After a thorough analysis of file generation behavior of the workflows, it became evident that the root cause of the server crash problems occur at the bursts in transfer requests. Our data movement service was originally implemented to deliver the data products as

promptly as possible, so asked RFT to transfer each file as soon as it received a request. As a result, the bursts in transfer requests get propagated all the way to the GridFTP servers and cause the servers to periodically crash. To solve this problem, the data movement service was enhanced with a flow control buffer algorithm to smooth out the file workloads sent to the RFT service by distributing across time the request bursts it receives. The algorithm also limits the maximum number of concurrent RFT transfers at any time based on a configurable approximation of the current resource limit of the GridFTP servers.

Fig. 5 shows the high-level pseudo code of our implementation. When our file movement service receives a transfer request, it stores the request in a

```
private TransferRequestQueue queue;
private int outstandingRequestCount;

// flow control buffer thread
public void run() {
    while (true) {
        // RESOURCE LIMIT is an approximation of the resource
        // limit of file transfer servers
        if (outstandingRequestCount < RESOURCE_LIMIT) {
            if (queue.size >= BATCHING_THRESHOLD ||
                queue.age >= AGE_THRESHOLD) {
                // Batch individual transfers into one RFT
                // request if there are enough entries to
                // make a batch or if entries have aged
                // beyond the threshold
                transferList = queue.remove(BATCHING_THRESHOLD);
                requestRFT(transferList);
                outstandingRequestCount++;
            } else {
                // wait until the number of outstanding requests
                // is lower than RESOURCE_LIMIT
                wait();
            }
        }

        public void receivedRequest(Request request) {
            queue.add(request);
        }

        public void rftFinishedCallback() {
            outstandingRequestCount--;
        }
    }
}
```

Fig. 5. Flow control buffer algorithm pseudo code

flow control buffer. A separate thread periodically checks the queue to determine when new RFT requests are to be sent, and thus effectively smoothes out any bursts in the incoming requests.

The algorithm has a counter that keeps track of the number of outstanding requests made to the RFT service, and compares this to a parameter of resource limit. This resource limit is a configurable service parameter making it easy to adapt to changes (i.e., increases) in the underlying GridFTP organization at a TeraGrid site.

To further increase performance, our algorithm takes advantage of the concurrent transfer capability built into RFT by combining several individual file transfers from the queue into a single RFT request. When the number of requests in the queue has reached the pre-determined batching factor, they are combined into a single request and sent to the RFT service. However, our algorithm also takes aging into consideration to

prevent starvation, so that even if the number of requests in the queue is below the batching threshold, they are still sent to the RFT service if they have aged beyond an amount of time regardless of the batching factor.

Simultaneous with our efforts, the Globus team identified a number of issues with their latest release of RFT that resulted in improved performance [14]. This version we understand will be officially released soon.

## 6 EXPERIMENTAL SETUP AND RESULTS

After integrating our solutions into the LEAD Science Gateway data movement and naming service, we run some experiments using the new RFT. In order to make a meaningful comparison with the data we measured earlier, we craft our experiment settings to faithfully match the original activities.

For the 723 file transfers from the captured logs, we extract the time information on when the requests were made, and create a test client script to reenact these transfers at the appropriate time with accuracy to the second. In addition, we look up the file sizes in our repository, and reproduce them with random bit patterns of the same sizes. However, since some of the original transfers failed and the files never made it to our repository, we have no way to tell their sizes. In such cases, we use a default size of 1GB. The log file used can be made available upon request for anyone interested in duplicating our experiment.

For our experiment, the source GridFTP server for file transfer is `gridftp1.bigred.teragrid.iu.edu:2812`, and the destination GridFTP server is `gridftp2.bigred.teragrid.iu.edu:2812`. This configuration is realistic because the majority of transfers captured in our log are from the scratch space to our data repository, which are both accessed from bigred. Our data movement service is hosted on `tyr12`, one of the 16-node cluster that each has dual 2.0GHz AMD Opteron processors and 16GB of RAM. Globus Toolkit with RFT v4.0.6 plus patch is also installed on `tyr12`, and our test client for launching the experiment is installed on `tyr02`.

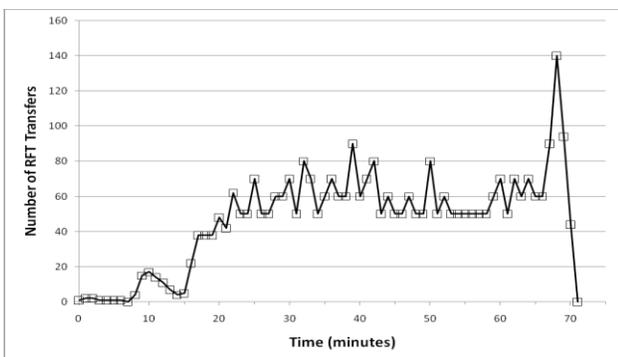


Fig. 6. Number of concurrent RFT transfers over time.

Fig. 6 plots the number of ongoing RFT transfers every minute during our experiment. From this graph, we can see that between the 20<sup>th</sup> and 60<sup>th</sup> minute mark, the number of concurrent transfers is contained mostly between 50 and 80, while originally, the number of transfers has a burst and increase to as high as 160 (please see Fig. 2).

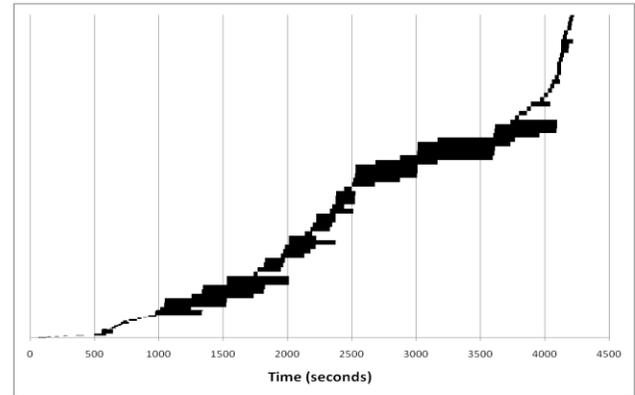


Fig. 7. Time span of RFT transfer requests.

However, in the last 5 minutes of the experiment, there is a spike on the graph indicating that the number of concurrent transfers reaches 140, and then immediately drops down towards 0 in the following few minutes. This pattern was not observed on the original data. To explain this, we need to look at this plot together with Fig. 7, which is a Gantt chart showing the transfer time span of all files in our experiment. Near the upper right corner of this graph, there is a long and thin tail indicating a large number of transfers started and finished quickly, all within these a few minutes. In addition, a few large file transfers started much earlier also spanned horizontally across into this time interval and finished within these few minutes. Since the number of concurrent transfers is calculated based on minutes, and does not take the number of finished transfers into account until the following minute, it makes Fig. 6 appear as if the number of concurrent transfers had a huge burst, although in reality, that number was well under threshold. In fact, during each of these minutes, the same number of transfers finished as there were number of requests made (which explains the rapid drop in the number of concurrent transfers). From the captured logs, we find most these small files are output from the Lateral Boundary Interpolator service, whose sizes are between 61MB and 86MB each. In addition, there were also a few dozen standard out and standard error files produced by the workflow, and these files are even smaller, only a few KB.

One interesting observation is that the experiment lasted for over 70 minutes, and many RFT transfer requests were made outside of the 1-hour time frame. This is because the flow control buffer in the data

movement service holds requests in its queue when the number of outstanding requests is at maximum. We argue this small delay is well outweighed by the gains in 100% reliability.

It is further worth noting that the Gantt chart from our experiment (Fig. 7) looks like “pixel blocks” as if it were blown up from a smaller bitmap image, yet the Gantt chart from the original data (Fig. 4) has a more “organic” look along the edges. The “pixel blocks” effect is a graphical confirmation of how the flow control buffer in the data movement service combines different transfers into a single RFT request, so that they all have the same starting and ending time.

## 7 CONCLUSIONS AND FUTURE WORK

Through smoothing bursts in file transfer workload and control for limited resources at the GridFTP servers, we were able to achieve a solution to the problem of file transfer workloads that overwhelm GridFTP servers and cause transfer failures. The current algorithm may keep transfer requests waiting in a queue longer than needed under heavy load, but this delay is offset by a substantial gain in reliability. That is, the workload from the original logs took 60 minutes to transfer 545 of the 723 workflow data files back to the personal workspace at a failure rate of 20%<sup>1</sup>. With the new smoothing solution, it took 70 minutes to transfer all 723 workflow data files back to the workspace with failure rate of 0%. In the future, this algorithm may undergo further improvements by leveraging knowledge of file sizes, so that files of similar sizes are combined together into one request, and also the number of transfers in a single RFT request can be dynamically determined by the total size of the files.

## ACKNOWLEDGEMENTS

The authors would like to thank John Bresnahan, Martin Feller, Dan Fraser, Ravi Madduri, and Stuart Martin from the Globus Team of Argonne National Lab, Derek Simmel from Pittsburgh Supercomputing Center, and Mike Lowe from IUPUI.

## REFERENCES

- [1] Allcock, W., J. Bester, et al., “Data management and transfer in high-performance computational grid environments”, *Parallel Computing*, 2001.
- [2] “Comprehensive file transfer service”, <http://gridport.net/services/cft/>
- [3] Coveney, P.V., M.J. Harvey, et al., “Development and deployment of an application hosting environment for grid based computational science”. In *Proceedings of the UK e-Science All Hands Meeting*, Sept. 2005.

- [4] “Lustre: a scalable, high-performance file system”, Cluster File Systems, Inc., Nov. 2002.
- [5] Madduri, R.K., C.S. Hood, and W.E. Allcock, “Reliable file transfer in grid environments”. In *Proceedings of the 7<sup>th</sup> Annual IEEE Conference on Local Computer Networks (LCN)*, 2002.
- [6] Plale, B., and D. Gannon, “Active management of scientific data”, *IEEE Internet Computing special issue on Internet Access to Scientific Data*, Vol. 9, No. 1, pp. 27-34, Jan/Feb 2005.
- [7] Schwan, P., “Lustre: building a file system for 1,000-node clusters”. In *Proceedings of the 2003 Linux Symposium*, July 2003.
- [8] Shirasuna, S., and D. Gannon, “Xbaya: a graphical workflow composer for the web services architecture”, *Technical Report 004*, LEAD, 2006.
- [9] Simms, S.C., G.G. Pike, and D. Balog, “Wide area filesystem performance using lustre on the teragrid”, *TeraGrid 2007*, Madison, WI. June 2007.
- [10] Sun, Y., S. Jensen, S.L. Pallickara, and B. Plale, “Personal workspace for large-scale data-driven computational experiment”, *7<sup>th</sup> IEEE/ACM International Conference on Grid Computing (Grid'06)*, Sept. 2006.
- [11] Thain, D., J. Basney, S.C. Son, and M. Livny, “The kangaroo approach to data movement on the grid”. In *Proceedings of the 10<sup>th</sup> IEEE International Symposium on High Performance Distributed Computing*, 2001.
- [12] “Trebuchet: a multi-scheme file-transfer API and client library”, <http://confluence.ncsa.uiuc.edu/display/MRDPUB/Trebuchet>
- [13] Droegemeier, K., V. Chandrasekar, et al., “Linked environments for atmospheric discovery (LEAD): architecture, technology roadmap and deployment strategy”, *21<sup>st</sup> Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*, Jan. 2005.
- [14] Fraser, D., I. Foster, et al., “Engaging with the LEAD science gateway project: lessons learned in successfully deploying complex system solutions on teragrid”, to appear in *TeraGrid 08 Conference*. Jun. 2008.

---

<sup>1</sup> Of the 723 original transfers, 34 started late in the hour and were only assumed to be successful since their outcomes were not captured in this hour-long log file, so the actual failure rate could have been even higher.