Supplementary Material: On Timely Staging of HPC Job Input Data

Henry M. Monti, *Student Member, IEEE,* Ali R. Butt, *Senior Member, IEEE,* and Sudharshan S. Vazhkudai

I. Related Work

In this section, we categorize and discuss several related works on HPC data management.

A. Current HPC Data Management

Typical HPC users either perform out-of-band manual staging that lacks coordination with job start-up, or include the staging commands in the job scripts which causes expensive stalls while waiting for data to be brought in. In either case, point-to-point data movement tools, e.g., `scp`, GridFTP [1], hsi [2], are used to move data. In contrast, the focus of Just in Time (JIT) staging is to orchestrate data staging to complete just when the computation is to begin, which has not been considered in prior works.

B. HPC Data Scheduling and Coordination

HPC job schedulers such as PBS Pro [3] and Moab [4] support data staging based on a computation schedule. However, these solutions do not adapt the data staging to changes in job startup times. There is no way to expedite the transfer as they only support point-to-point transfer protocols. Consequently, these solutions cannot address network volatility either. Stork [5] can handle network vagaries. However, Stork is designed for grid environments, and while complementary, does not directly address HPC data management design-space that is the focus of this paper.

Moreover, BAD-FS [6] coordinates input data and computation by exposing distributed file system decisions to an external workload-aware scheduler. While we have the same common goal as this work, our approach aims to inherently improve the job workflow without creating a new file system.

Finally, we have also examined different aspects of data scheduling and coordination in our prior work: decoupling the staging of a job's data from its execution by creating separate data and batch queues [7]; and treating the HPC center's scratch space as an integrated cache [8]. However, what is still needed is the ability to bring the data into the HPC center's scratch space in a timely fashion, which we build in this work.

C. HPC Data Movement

DMOVER [9] tool is used for moving data in the TeraGrid [10] by aggregating data transfer commands in a script and scheduling them using a separate queue. However, in contrast to our decentralized JIT staging DMOVER only addresses point-to-point data transfers using GridFTP. Kangaroo [11] employs intermediate storage to provide reliability against transient resource availability in grid environments. However, it simply provides a staged transfer mechanism and does not handle network dynamicity.

IBP [12] offers a data distribution infrastructure with a set of strategically placed "storage depots." In contrast, we combine both a staged and decentralized data delivery. The induction of user-specified or cloud storage nodes allows us to optimize the staging on a per-user basis, which is not possible with IBP. Further, our approach is unique as we strive to meet a potentially volatile job start up time, despite network variability. IBP has been extended [13] with strategies to reconcile the rate of data production with that of available network resources. This is complementary to our work.

The GridFTP overlay network service [14] implements a specialized data storage interface (DSI) to achieve split-TCP functionality. In [15], the authors have extended this effort to use previous transfers as a measure of when to use a particular node in the transfer overlay. Our work differs as it delivers data on a deadline (job start time) and further uses dynamic measurements to adapt and adjust the fan-in of transfers.

The approach of downloading large files from several mirror sites has been validated by its wide-spread use in BitTorrent [16], and many other protocols have been proposed [17], [18], [19]. These works are complementary, and we build on their principles, especially BitTorrent.

Finally, the Network Weather Service (NWS) [20] provides a powerful framework that allows the resources of distributed computers to be monitored. We use NWS measurements to determine a path within a network of nodes and dynamically adjust it based on bandwidth degradation.

II. Additional Design Concerns

In this section, we provide additional discussion regarding the end-users' motivation for participating in the collaborative staging process and also present some alternative design considerations.

1) Motivation for Collaboration: In today's HPC environment, supercomputing jobs are almost always collaborative in nature. In fact, a quick survey of jobs awarded compute time on the ORNL NLCF, through the DOE's INCITE [21] program, suggests that these jobs involve multiple users from multiple institutions. This collaborative property is even more true in TeraGrid [10], where jobs are usually from a *virtual*

Henry Monti and A.R. Butt are with the Department of Computer Science, Virginia Tech, Blacksburg, VA 24061. E-mail: {hmonti,butta}@cs.vt.edu.

Sudharshan S. Vazhkudai is with Oak Ridge National Laboratory, Oak Ridge, TN 37831. E-mail: vazhkudaiss@ornl.gov.

organization, which is a set of geographically dispersed users from different sites, coming together to solve a problem of mutual interest for a certain duration. A example of this use-case is the Earth System Grid [22], where it is not uncommon for different research groups to voluntarily replicate climate model data. In such cases, it is clear that many users, from different sites will be interested in seeing the job run to completion, with as little delay as possible. This emerging property of collaborative science can be exploited to perform a collaborative staging of job input data. We therefore argue that there exists a *natural incentive* to provide resources for the JIT staging process, and that such resources are essential to the application itself and should not be construed as an "extra" component needed solely for JIT staging.

2) Impact on Infrastructure Costs: We reiterate that our design does not require the explicit setup and management of cloud, landmark, and intermediate nodes. Instead, it leverages and "piggybacks" on *existing* infrastructure. Several national testbeds, e.g., TeraGrid [10], REDDNET [23], etc., are already in production and can act as such nodes, without incurring any additional costs such as electricity, manpower, and management costs. Moreover, intermediate nodes use resources that are already part of the "collaborative" job. We also do not require extra provisioning of network bandwidth, rather employ the residual bandwidth that would have otherwise gone unused. Nonetheless, extra usage, if necessary, can be construed as necessary for completing of the collaborative job, and the burden can be shared by all the collaborators, not unlike when researchers have to utilize extra resources individually to support a demanding job. Overall, our design also achieves better utilization of resources and possibly a higher system-wide efficiency.

3) Alternative Data Staging Designs: There are several possible alternative solutions for the HPC staging problem, namely, adding more scratch space, streaming data directly and not using the scratch space, and moving computations closer to data. In the following, we discuss why we did not adopt these options in our design.

First, we reiterate that simply adding more scratch is not practical (Section I in the main paper), as scratch is a precious commodity and provisioning more scratch means taking dollars away from buying FLOPS, and more FLOPS are how most HPC acquisition proposals are won.

Second, streaming data online and bypassing scratch to support HPC applications is not viable and sustainable (based on Top500 supercomputers). Additionally, distributed filesystems or middleware are seldom an option for extreme-scale, leadership class machines. The scratch space is a parallel file system that is made available at a mount point, to the hundreds of thousands of compute cores where the parallel job's processes run. Serving the hundreds of thousands of compute processes of a currently running job through remote I/O to a distributed file system that is geographically dispersed is a significantly expensive option, and an impractical one. Furthermore, streaming mechanisms cannot match the I/O rates required to keep such large systems busy, e.g., Jaguar [24] scratch offers I/O rates of 256 GB/s.

Third, moving computation closer to data is a compelling idea, but there are numerous HPC applications, e.g., DOE supercomputer and NSF TeraGrid applications, which cannot be sustained on users' local clusters where data may be available. Our design takes all these factors into consideration for realizing a practical solution to the staging problem.

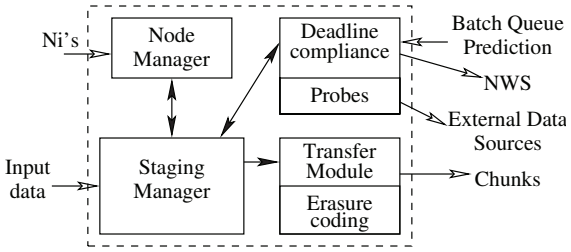


Fig. 1. Implementation architecture for timely staging.

III. Implementation

We have implemented the JIT staging manager using about 3500 lines of C code. Figure 1 shows the overall architecture as well as the interactions between the manager components.

a) Integration with Job Submission: To facilitate easy adoption of our scheme by the community, we have integrated it with the widely-used PBS [25] job submission system. Specifically, we have instrumented the job submission scripts to let users specify intermediate nodes and deadlines. An example instrumented PBS script is shown in Figure 2, where the user specifies intermediate nodes and deadlines as well as details such as available storage capacities. The nodes listed in the script are just a suggestion, and the actual runtime queries these nodes directly for availability as needed.

The annotated script is submitted to the staging manager on the center, which filters out the staging-specific directives and forwards the remaining script to the standard batch queue, but with a dependency on the staging task. We extend our earlier works [7], [8] on instrumenting the job submission system for this purpose.

b) Integration with BitTorrent and NWS: We exploit BitTorrent's [16] scatter-gather protocol for transferring data by extending the protocol to use NWS bandwidth measurements. The NWS measurements are integrated with BitTorrent to dynamically select fast locations where a particular dataset can be retrieved, and adapt to changing network behavior by adjusting fan-in to enable staging of data in time.

Since our system uses BitTorrent, the source only needs to send one copy of the data to the intermediate nodes. Once

```
#PBS -N myjob
#PBS -l nodes=128, walltime=12:00
mpirun -np 128 ~/MyComputation
#Stage file://SubmissionSite:/home/user/input1
        file:///home/scratch/user/input1
#Stage wget://WebRepo:/input2
        file:///home/scratch/user/input2
#InterNode node1.Site1:49665:50GB
#InterNode nodeN.SiteN:49665:30GB
#JobStartDeadline 11/14/2011:12:00
```

Fig. 2. An instrumented PBS script for timely staging.

complete, if bandwidth is a consideration the source can stop "seeding", and the intermediate nodes will propagate data among themselves. However, if the source stays online after the "client offload" completes, the transfer could be quicker. Additionally, the HPC center will only need to pull one copy of the data to complete the staging process.

c) Center-wide Global Staging Considerations: Since we anticipate that all jobs, along with their staging needs, will be submitted through the staging manager, we have instrumented into the manager certain global optimizations that can be performed across all jobs. (1) All jobs that desire a staging to the Level-N, i.e., one hop away from the center, can be started immediately. Since these staging operations do not use any center resources — neither occupying scratch space nor consuming bandwidth — the data can be brought closer to the center and pulled in much faster when needed. (2) A job whose startup deadline tightens during the course of a previously initiated staging will be given higher priority if it is determined that the staging may not complete in time. For instance, this could mean providing more flows to maximize the last leg of the transfer, using more of the center's in-coming bandwidth.

d) Ensuring Data Reliability: To ensure that data is reliably staged on the center, we employ replication of data by sending out chunks to more than a single location. This is a tunable parameter in our implementation and users can specify the minimum number of replicas that should be created for a given dataset. If necessary, more space-efficient erasure codes can be used. The erasure code that we have used in our implementation is Reed-Solomon [26] in 4:5 coding configuration, i.e., four input chunks are coded to produce five output chunks, with a redundancy of 25%. The chunk-size is also a tunable parameter which can be set based on the size of the datasets being transferred.

e) Multi-Input Staging: Our implementation is capable of retrieving data from more than a single source, directly as well as incorporating it into the decentralized transfer. The data sources are provided as links in the job-submission script. If the external data source runs an instance of our software, the staging manager can simply use the NWS information to decide between direct or decentralized staging. However, if the external source does not support NWS (e.g., the cloud), the staging manager uses small scale tests, e.g., a partial download from a web repository, to determine expected transfer times and make staging decisions. In this case, the goal of the staging manager is to ensure staging of all input data from all sources before the predicted job startup time.

IV. Simulating HPC Data Staging Process

To systematically study the staging process in detail, we have developed a realistic simulator, *simStagein*, which models both job execution and data staging.

a) Job scheduling: In *simStagein*, jobs are scheduled using a First-Come First-Served (FCFS) policy with back-filling that is common in HPC centers [24]. The goal of this scheduler is to strike a balance between potential idle cores and the HPC center's desire to cater to "hero apps" that could take up an entire supercomputer.

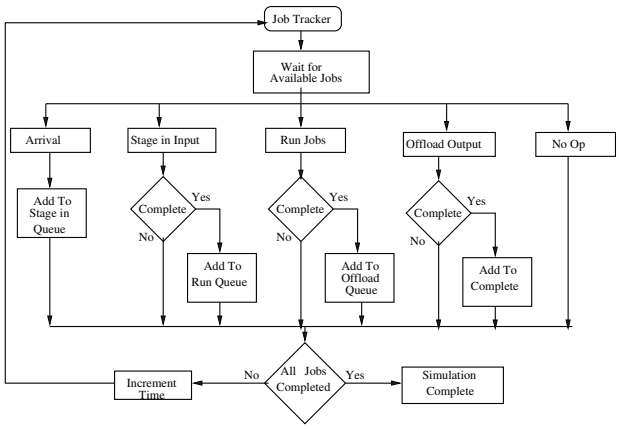


Fig. 3. Control flow in *simStagein*.

b) Job traces: simStagein utilizes a number of different traces to provide an accurate model of the system. The job traces were obtained from ORNL's Jaguar supercomputer [24] and represent nearly three years of job execution [8]. These traces provide for each job: arrival time, start time, total job execution time, and the compute and memory resources used. This information is used to estimate the amount of data consumed by a job [27]. Each job in the trace corresponds to a job staging-in and executing in our simulator.

c) Bandwidth traces: We model the intermediate nodes by using NWS pairwise bandwidth measurements from 50 different sites on PlanetLab [28] collected over a duration of 96 hours. Each simulated node in *simStagein* is assigned a measured trace. Depending on the number of nodes and duration of the simulation a trace may be assigned to multiple nodes and a node may loop through its associated trace. If a node is used for multiple stagings at the same time, the bandwidth is equally divided between the stagings.

d) Simulator output: simStagein provides an output trace with information about overall scratch space usage and the time it would take to stage the required data for a given job. This information can then further be used to determine any delay in meeting job scheduling deadlines.

e) Flow of control in simStagein: simStagein maintains a pool of randomly selected nodes arranged in a configurable topology to use as intermediate nodes. Moreover, *simStagein* can also capture varying storage capacities of the nodes and can alter staging paths based on the capacities.

Figure 3 illustrates *simStagein*'s operation. The main driver is a *Job tracker* that reads the logs, and selects an appropriate action for the simulator to take. We have opted for using the same time-scale as the logs. At each job arrival, the tracker places it in a *wait queue*. The job input data staging is then started. The staging process may take many simulator ticks depending on the size of the input data, but once the process completes the job is moved to a *run queue*. The job will wait there until sufficient compute resources become available. Once the job completes its execution, it moves to the offload queue. If the simulator is modeling a decentralized offload [29], intermediate nodes will be chosen and the offload process will begin. If the standard approach is used, the data

TABLE I
THE TIME TO TRANSFER A 2 GB FILE USING STANDARD BITTORRENT. THE EQUIVALENT PHASES FOR OUR SCHEME ARE SHOWN IN BRACKETS.

Phase	Time (s)
Send to intermediate nodes (Client Offload)	1428
Download at HPC center (Center Pull)	362

will remain on the scratch until it is purged by the center. Finally, *simStagein* also provides accounting and statistics about the offload process, such as the scratch space used and the data read, as well as other vital statistics.

V. ADDITIONAL RESULTS

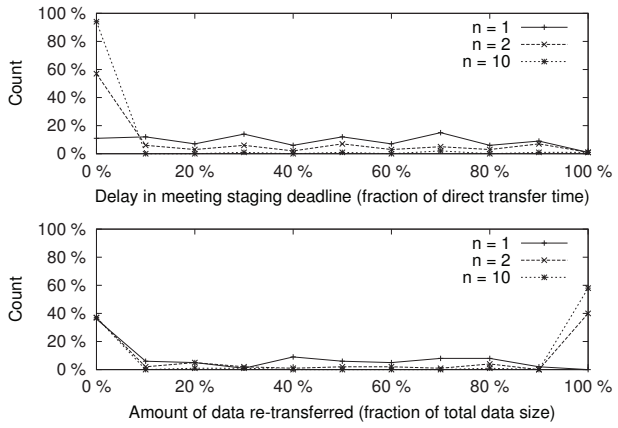
In this section, we present additional results of interest from experiments using both our implementation and log analysis.

1) Effect of Using NWS Measurements: First, we compare our NWS-based monitored transfer approach with a standard BitTorrent-based data transfer. In this case, we use NWS bandwidth measurements to greedily provision Level-2 nodes to increase the fan-in, i.e., the number of nodes simultaneously transferring data to the center, to utilize the maximum center in-bound bandwidth. Table I shows the times taken to deliver a 2.0 GB file using the standard BitTorrent protocol. Compare these to the transfer times using our timely staging shown in the main paper in Table II: both Client Offload and Center Pull in our approach out-perform by 11.5% and 6.8%, respectively, the corresponding steps in regular BitTorrent transfer. These results show that active bandwidth monitoring serves as a good technique to improve staging times.

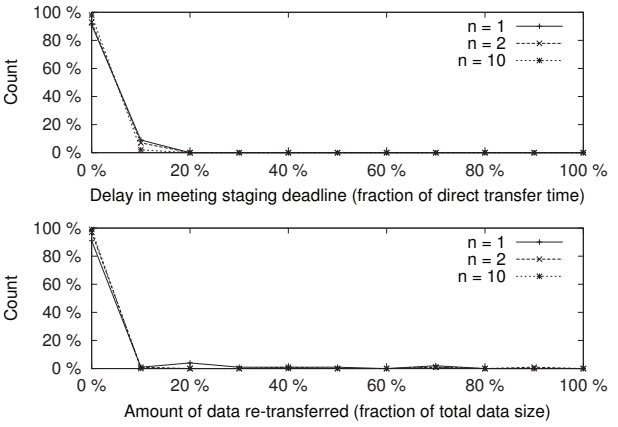
2) Employing Decentralized Staging: In the experiments presented in the main paper, the bandwidth available between the Level-2 nodes and the center, which dictates Center Pull times, is greater than that between the client and the center, which dictates direct transfer time. Thus, the center always decided to perform decentralized staging. In this next experiment, we modified the setup to use a faster node as the client site, and repeated the experiment for staging a 2 GB file. First, we do the transfer without considering direct transfer and always using decentralized staging. Second, we repeat the experiment with the ability to choose between direct and decentralized staging depending on the ability to meet a transfer deadline (job startup). We observed that for the first case, the time to stage and transfer the data to the center was 2867 seconds. In contrast, for the second case the direct transfer completed in 968 seconds, an improvement of 66.2%. This stresses the need for the staging mechanisms to dynamically adjust to the variations in the system behavior, and to not be hard-wired to simply always do a staged transfer or a direct transfer.

3) Behavior Under Failures: We also examine how failure in the scratch space affects the ability of a transfer scheme to meet a given job deadline. Here, we capture the early-transferring approach of users by starting the direct transfers as early as $T_{JobStartup} - n * T_j$, with $1 \leq n \leq 10$. Next, we randomly introduce a single failure on the scratch space between the time of starting the transfer and $T_{JobStartup}$, and determine the delay in meeting the job deadline, as well as the extra amount of data that has to be transferred. For timely staging, we assume perfect prediction, so it starts staging-in data as late as possible for a given file size. The experiment is repeated 25 times using files of sizes from 1 GB to 5 GB, for each studied n. Figure 4 shows the distribution of delay in meeting a deadline and the amount of data re-transferred, respectively. In the distributions, a higher count for a smaller x-axis value is desirable as that implies less delay and higher chances of meeting a deadline, and less data re-transfers. Our timely staging shows excellent properties with 98% of the transfers completing with no delay. In contrast, only a direct transfer that starts as early as with $n = 10$ is able to come close with 94% transfers without delay. With $n = 2$, only 31% of direct transfers complete in time. The flip side is that by staging early, the data remains exposed to the failures on the scratch and possible re-transfers. It is observed that while over 91% of the transfers in our approach had no retransmissions due to exposure to failures, that is only true for 36% of the cases with direct transfers.

Note that since we introduce a single failure, the maximum overhead is 100%. In real scenarios, multiple failures can further exacerbate the problem, as the re-transfer may now take much longer than the earlier transfer or failures in the system



(a) Direct transfer.



(b) Timely staging.

Fig. 4. The distribution of staging delay and re-transmission overhead for 25 transfers with one scratch space failure. n represents by how early data staging is started before job startup, with higher n implying an earlier start of staging process.

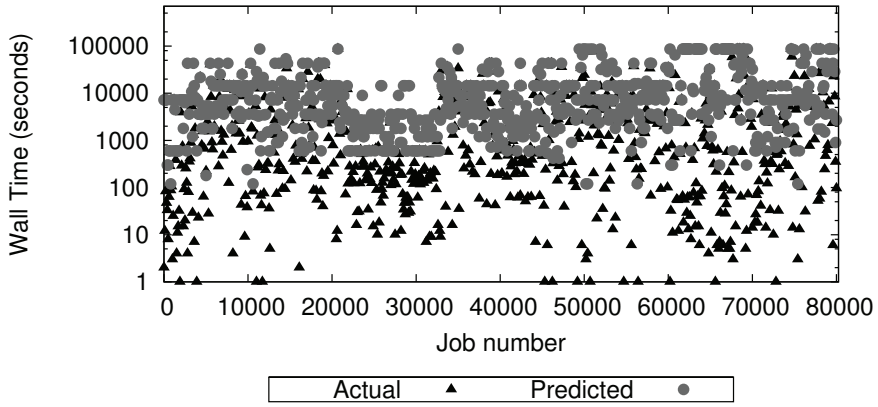


Fig. 5. The actual and user predicted run-times for each job in the logs. Users typically request significantly more wall time than necessary.

may prevent immediate response to a failure. This implies that delaying staging is preferable. Thus, JIT staging is able to withstand failures much closer to the job deadline, and the delay if any is small, and can be mitigated by assuming a slightly tighter deadline than actual (Section II main paper).

4) Log Analysis: We examine the accuracy of user-estimated run-times, as many works [30] have noted that users generally request more resources than required by their jobs. Figure 5 (a larger version of Figure 5 from the main paper) plots the user requested run-times with the actual run-times as recorded in the logs, and confirms this perception. Across the logs, the users over-estimated the requirements by 50.9 times on average for jobs longer than 30 seconds (430 times for all jobs), mostly due to jobs ending prematurely.

REFERENCES

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link. The Globus Striped GridFTP framework and server. In *Proc. Supercomputing*, 2001.

[2] M. Gleicher. HSI: Hierarchical storage interface for HPSS. http://www.hpss-collaboration.org/hpss/HSI/.

[3] Pbs pro technical overview: Scheduling and file staging. https://secure.altair.com/sched_staging.html.

[4] Cluster resources inc. http://www.clusterresources.com/.

[5] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the grid. In *Proc. ICDCS*, 2004.

[6] J. Bent, D. Thain, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Explicit control in a BADFS. In *Proc. NSDI*, 2004.

[7] Z. Zhang, C. Wang, S. S. Vazhkudai, X. Ma, G. Pike, J. Cobb, and F. Mueller. Optimizing center performance through coordinated data staging, scheduling and recovery. In *Proc. SC*, 2007.

[8] Henry Monti, Ali R. Butt, and S. S. Vazhkudai. /Scratch as a Cache: Rethinking HPC Center Scratch Storage. In *Proc. ACM ICS*, 2009.

[9] DMOVER: Scheduled data transfer for distributed computational workflows. http://www.psc.edu/general/software/packages/dmover/, 2008.

[10] Grid Infrastructure Group. TeraGrid. http://www.teragrid.org/.

[11] D. Thain, S. Son J. Basney, and M. Livny. The kangaroo approach to data movement on the grid. In *Proc. HPDC*, 2001.

[12] J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the network. In *Proc. Network Storage Symposium*, 1999.

[13] Viraj Bhat, Scott Klasky, Scott Atchley, Micah Beck, Doug Mccune, and Manish Parashar. High performance threaded data streaming for large scale simulations. In *Proc. Grid*, 2004.

[14] P. Rizk, C. Kiddle, and R. Simmonds. A gridftp overlay network service. In *Proc. Grid* , 2007.

[15] G. Khanna, U. Catalyurek, T. Kurc, R. Kettimuthu, P. Sadayappan, I. Foster, and J. Saltz. Using overlays for efficient data transfer over shared wide-area networks. In *Proc. Supercomputing*, 2008.

[16] Bram Cohen. BitTorrent Protocol Specification, May 2007. http://www.bittorrent.org/protocol.html.

[17] P. Rodriguez, A. Kirpal, and E. W. Biersack. Parallel-access for mirror sites in the internet. In *Proc. IEEE Infocom*, 2000.

[18] James S. Plank, Scott Atchley, Ying Ding, and Micah Beck. Algorithms for high performance, wide-area distributed file downloads. *Parallel Processing Letters*, 13(2):207–224, 2003.

[19] Rebecca L. Collins and James S. Plank. Downloading replicated, wide-area files – a framework and empirical evaluation. In *Proc. 3rd IEEE International Symposium on Network Computing*, 2004.

[20] Rich Wolski, Neil Spring, and Jim Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Gen. Computing Systems*, 15(5):757–768, 1999.

[21] Department of Energy, Office of Science. Innovative and Novel Computational Impact on Theory and Experiment (INCITE), Jan 2008. http://www.er.doe.gov/ascr/incite/.

[22] Earth system grid. http://www.earthsystemgrid.org, 2006.

[23] Reddnet enabling data intensive science in the wide area, 2008. http://www.reddnet.org/mwiki/index.php/Main_Page.

[24] National Center for Computational Sciences. http://www.nccs.gov/.

[25] A. Bayucan, R.L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten. Portable Batch System: External reference specification. Nov. 1999. http://www-unix.mcs.anl.gov/openpbs/docs/v2_2_ers.pdf.

[26] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, 1997.

[27] Henry Monti, Ali R. Butt, and Sudharshan S. Vazhkudai. Reconciling Scratch Space Consumption, Exposure, and Volatility to Achieve Timely Staging of Job Input Data. In *Proc. IPDPS*,2010.

[28] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. ACM HotNets*, 2002.

[29] Henry Monti, Ali R. Butt, and Sudharshan S. Vazhkudai. Timely offloading of result-data in hpc centers. In *Proc. 22nd ACM ICS*, 2008.

[30] Wei Tang, Narayan Desai, Daniel Buettner, and Zhiling Lan. Analyzing and adjusting user runtime estimates to improve job scheduling on the blue gene/p. In *Proc. IPDPS*, 2010.