Stork data scheduler: mitigating the data bottleneck in e-Science

BY TEVFIK KOSAR1,3,*, MEHMET BALMAN2, ESMA YILDIRIM1, SIVAKUMAR KULASEKARAN3 AND BRANDON ROSS3

1Department of Computer Science and Engineering, State University of New York, Buffalo, NY, USA
2Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA
3Center for Computation and Technology, Louisiana State University, Baton Rouge, LA, USA

In this paper, we present the Stork data scheduler as a solution for mitigating the data bottleneck in e-Science and data-intensive scientific discovery. Stork focuses on planning, scheduling, monitoring and management of data placement tasks and application-level end-to-end optimization of networked inputs/outputs for petascale distributed e-Science applications. Unlike existing approaches, Stork treats data resources and the tasks related to data access and movement as first-class entities just like computational resources and compute tasks, and not simply the side-effect of computation. Stork provides unique features such as aggregation of data transfer jobs considering their source and destination addresses, and an application-level throughput estimation and optimization service. We describe how these two features are implemented in Stork and their effects on end-to-end data transfer performance.

Keywords: data-intensive computing; input/output scheduling; throughput optimization; e-Science; Stork

1. Introduction

Scientific applications generate increasingly large amounts of data, often referred as the ‘data deluge’ [1], which necessitates collaboration and sharing among the national and international research institutions. Simply purchasing high-capacity, high-performance storage systems and adding them to the existing infrastructure of the collaborating institutions does not solve the underlying and highly challenging data-handling problem. Scientists are often forced to spend a great deal of time and energy on solving basic data-handling issues, such as the physical location of data, how to access it and/or how to move it to visualization and/or compute resources for further analysis. The systems

*Author for correspondence (tkosar@buffalo.edu).

One contribution of 12 to a Theme Issue ‘e-Science: novel research, new science and enduring impact’.
managing these resources must provide robust scheduling and allocation of network and storage resources, as well as optimization of end-to-end data transfers over wide area networks (WANs).

According to the *Strategic Plan for the US Climate Change Science Programme* [2], one of the main objectives of the future research programmes should be ‘enhancing the data management infrastructure’, since ‘The users should be able to focus their attention on the information content of the data, rather than how to discover, access, and use it’. [2, pp. 143–150]. This statement by the CCSP summarizes the goal of many cyberinfrastructure efforts initiated by the US Department of Energy (DOE), National Science Foundation (NSF) and other federal agencies, as well as the research direction of several leading academic institutions. This is also the main motivation for the work presented in this paper.

Traditional distributed computing systems closely couple data handling and computation. They consider data resources as second-class entities, and access to data as a side-effect of computation. Data placement (i.e. access, retrieval and/or movement of data) either is embedded in the computation and causes the computation to delay or is performed by simple scripts that do not have the same privileges as compute jobs. The inadequacy of traditional distributed computing systems in dealing with complex data-handling problems in our new data-rich world has motivated a new paradigm called *data-aware distributed computing* [3].

In previous work, we have introduced the concept that the data placement activities in a distributed computing environment need to be first-class entities just like computational jobs, and presented the first batch scheduler specialized in data placement and data movement: Stork [4]. This scheduler implements techniques specific to queuing, scheduling and optimization of data placement jobs, and provides a level of abstraction between the user applications and the underlying data transfer and storage resources. Stork is considered one of the very first examples of ‘data-aware scheduling’ and has been very actively used in many e-Science application areas including coastal hazard prediction and storm surge modelling [5]; oil flow and reservoir uncertainty analysis [6]; digital sky imaging [7]; educational video processing and behavioural assessment [8]; numerical relativity and black hole collisions [9]; and multi-scale computational fluid dynamics [10].

Our previous work in this area has also been acknowledged by the strategic reports of federal agencies. The DOE Office of Science report on data management challenges defines data movement and efficient access to data as two key foundations of scientific research [11]. The DOE report says: ‘In the same way that the load register instruction is the most basic operation provided by a CPU, so is the placement of data on a storage device. ... It is therefore essential that at all levels data placement tasks be treated in the same way computing tasks are treated’, and refers to our previous work [4]. The same report also states that ‘Although many mechanisms exist for data transfer, research and development is still required to create schedulers and planners for storage space allocation and the transfer of data’ [11, pp. 55–56].

In this paper, we elaborate on how a data scheduler like Stork can help to mitigate the large-scale data management problem for e-Science applications. Section 2 presents the related work in this area, and §3 gives background information about the Stork data scheduler. In the following sections, we describe...
two unique features of Stork: aggregation of data transfer jobs considering their source and destination addresses (§4) and the application-level throughput estimation and optimization service (§5). We conclude the paper in §6.

2. Background

Several previous studies address data management for large-scale applications [12–20]. However, scheduling of data storage and networking resources and optimization of data transfer tasks has been an open problem.

In an effort to achieve reliable and efficient data placement, high-level data management tools such as the reliable file transfer (RFT) service [21], the lightweight data replicator (LDR) [22] and the data replication service (DRS) [23] were developed. The main motivation for these tools was to enable byte streams to be transferred in a reliable manner, by handling possible failures such as dropped connections, machine reboots and temporary network outages automatically via retrying. Most of these tools are built on top of the Grid file transfer protocol (GridFTP) [14,15], which is a secure and reliable data transfer protocol especially developed for high-bandwidth WANs.

Beck et al. [24] introduced logistical networking that performs global scheduling and optimization of data movement, storage and computation based on a model that takes into account all the network’s underlying physical resources. Systems such as the storage resource broker (SRB) [25], the integrated rule oriented data system (iRODS) [26] and the storage resource manager (SRM) [27] were developed to provide a uniform interface for connecting to heterogeneous data resources and accessing replicated datasets.

Thain et al. [28] introduced the batch-aware distributed file system (BAD-FS), which was followed by a modified data-driven batch scheduling system [29]. Their goal was to achieve data-driven batch scheduling by exporting explicit control of storage decisions from the distributed file system to the batch scheduler. Using some simple data-driven scheduling techniques, they have demonstrated that the new data-driven system can achieve better throughput both over current distributed file systems such as the Andrew file system as well as over traditional central processing unit (CPU)-centric batch scheduling techniques, which are using remote inputs/outputs (I/O).

According to Stockinger [30] and Stockinger et al. [31], the entire resource selection problem requires detailed cost models with respect to data transfer. A cost model for data-intensive applications is presented in Stockinger et al. [32] where theoretical models for data-intensive job scheduling are discussed. In that work, a cost model is created that can determine whether it is more efficient to transfer the data to a job or vice versa. The metric for measuring efficiency is the effective time seen by the client application. The model includes all important factors in a distributed data Grid and takes various storage and access latencies into account to determine optimal data access. More general performance engineering approaches are discussed in Stockinger et al. [31]. In that work, they analyse a typical Grid system and point out performance analysis aspects in order to improve the overall job execution time of the system. Their focus is on the performance issues regarding data and replica management.
The studies that try to find the optimal number of streams for data scheduling are limited and they are mostly based on approximate theoretical models [33–36]. They all have specific constraints and assumptions. Also the correctness of the model is proved with simulation results mostly. Hacker et al. [34] claim that the total number of streams behaves like one giant stream that transfers in the capacity of the total of each stream’s achievable throughput. However, this model only works for uncongested networks. Thus it is not able to predict when the network will be congested. Another study [33] declares the same theory but develops a protocol which at the same time provides fairness. Lu et al. [35] model the bandwidth of multiple streams as a partial second-order polynomial equation and need two different throughput measurements of different stream numbers to predict the others. In another model, the total throughput always shows the same characteristics [36] depending on the capacity of the connection as the number of streams increases and three streams are sufficient to get a 90 per cent utilization. None of the existing studies are able to accurately predict the optimal number of parallel streams for best data throughput in a congested network.

3. Stork data scheduler

The Stork data scheduler [24,37,38] implements techniques specific to queuing, scheduling and optimization of data placement jobs, provides high reliability in data transfers and creates a level of abstraction between the user applications and the underlying data transfer and storage resources (including file transfer protocol (FTP), hypertext transfer protocol (HTTP), GridFTP, SRM, SRB and iRODS) via a modular, uniform interface. Stork is considered one of the very first examples of ‘data-aware scheduling’ and has been very actively used in many e-Science application areas, including: coastal hazard prediction, reservoir uncertainty analysis, digital sky imaging, educational video processing, numerical relativity and multi-scale computational fluid dynamics resulting in breakthrough research [5–10,38–40].

Using Stork, the users can transfer very large datasets via a single command. The checkpointing, error recovery and retry mechanisms ensure the completion of the tasks even in the case of unexpected failures. Multi-protocol support makes Stork a very powerful data transfer tool. This feature allows Stork not only to access and manage different data storage systems, but also to be used as a fall-back mechanism when one of the protocols fails in transferring the desired data. Optimizations such as concurrent transfers, parallel streaming, request aggregation and data fusion provide enhanced performance compared with other data transfer tools. The Stork data scheduler can also interact with higher level planners and workflow managers for the coordination of compute and data tasks. This allows the users to schedule both CPU resources and storage resources asynchronously as two parallel universes, overlapping computation and I/O.

Our initial end-to-end implementation using Stork consists of two parallel universes: a data subsystem and a compute subsystem. These two subsystems are complementary, the first specializing in data management and scheduling and the latter specializing in compute management and scheduling. The orchestration of these two parallel subsystems is performed by the upper layer workflow planning
and execution components. In cases where multiple workflows need to be executed on the same system, users may want to prioritize among multiple workflows or make other scheduling decisions. This is handled by the workflow scheduler at the highest level. An example for such a case would be hurricanes of different urgency levels arriving at the same time.

When integrating data placement into end-to-end workflow planning and management systems, we make use of similarities to instruction pipelining in microprocessors, where the life cycle of an instruction consists of steps such as fetch, decode, execute and write. A distributed workflow system can be viewed as a large pipeline consisting of many tasks divided into sub-stages, where the main bottleneck is remote data access/retrieval owing to network latency and communication overhead. Just as pipelining techniques are used to overlap different types of jobs and execute them concurrently while preserving the task sequence and dependencies, we can order and schedule data movement jobs in distributed systems independent of compute tasks to exploit parallel execution while preserving data dependencies. For this purpose, we expand the scientific workflows to include the necessary data-placement steps, such as stage-in and stage-out, as well as other important steps that support data movement, such as allocating and de-allocating storage space for the data and reserving and releasing the network links.

We have done a preliminary implementation of data-aware workflow management where data-awareness components were added to the workflow planning tool Pegasus and workflow execution tool Condor DAGMan. This preliminary implementation was used in coastal hazard prediction [5] and reservoir uncertainty analysis [6] applications. We have developed a preliminary model to choose between remote I/O and staging for particular applications. As one of the results of that work, we suggest that, for remote I/O to be more efficient than staging, the following equation should hold true:

\[ N_r - N_s < W_l + R_l, \]  

(3.1)

where \( R_l \) is the time to read from local disk to local memory, \( W_l \) is the time to write from local memory to local disk, \( N_s \) is the time to send data over the network via a staging protocol and \( N_r \) is the time to send data over the network via a remote I/O protocol. This equation shows that the time difference coming from using a specialized data transfer protocol versus a remote I/O protocol should be less than the overhead of extra read/write to the disk in staging. In other words, if your remote I/O library performs well in data transfer over the network, or your local disk performance is slow, remote I/O might be advantageous over staging. Otherwise, the staging method would perform better. One challenge would be estimating these parameters without actually running the application.

For this purpose, the Stork data scheduler includes network and storage bandwidth-monitoring capabilities that collect statistics on the maximum available end-to-end bandwidth, actual bandwidth utilization, latency and the number of hops to be travelled. In order to estimate the speed at which data movement can take place, it is necessary to estimate the bandwidth capability at the source storage system, at the target storage system and the network in between.
In the next two sections, we will present two unique features of Stork: aggregation of data transfer jobs considering their source and destination addresses and the application-level throughput estimation and optimization service.

4. Request aggregation

We have implemented the aggregation of data transfer requests in the Stork data scheduler in order to minimize the overhead of connection set-up and tear-down for each transfer. Inspired by prefetching and caching techniques in microprocessor design, data placement jobs are combined and embedded into a single request in which we increase the overall performance especially for transfers of datasets with small file sizes.

The Stork data scheduler accepts multiple data transfer jobs in a non-deterministic order. For each transfer operation, we need to initialize the network transfer protocol and set up a connection to a remote data transfer service. A single transfer $t$ consists of $t_{\text{setup}}$, in which we initialize the transfer module and prepare a connection to transfer the data, and $t_{\text{transfer}}$, in which we transfer data using the data transfer protocol over a network. Although this connection time is small compared with the total duration spent for transferring the data, it becomes quite important if there are hundreds of jobs to be scheduled. We have minimized the load put by $t_{\text{setup}}$ by multiple transfer jobs. Instead of having separate set-up operations such as $t_1 = t_{\text{setup}} + t_{\text{transfer}}$, $t_2 = t_{\text{setup}} + t_{\text{transfer}}$, ..., $t_n = t_{\text{setup}} + t_{\text{transfer}}$, we aggregate multiple requests to improve the total transfer time $t = t_{\text{setup}} + t_{\text{transfer}} + t_{\text{transfer}} + \cdots + t_{\text{transfer}}$.

Aggregating data placement jobs and combining data transfer requests into a single operation also has its benefits in terms of improving the overall scheduler performance, by reducing the total number of requests that the data scheduler needs to execute separately. Instead of initiating each request one at a time, it
Table 1. Request aggregation and the overhead of connection time in data transfers. $f_{num}$, number of data files sent in a single transfer operation.

<table>
<thead>
<tr>
<th>$f_{num}$</th>
<th>time (s)</th>
<th>thrpt (Mbps)</th>
<th>time (s)</th>
<th>thrpt (Mbps)</th>
<th>time (s)</th>
<th>thrpt (Mbps)</th>
<th>time (s)</th>
<th>thrpt (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.59</td>
<td>6.29</td>
<td>0.39</td>
<td>25.64</td>
<td>9.81</td>
<td>10.19</td>
<td>2.4</td>
<td>41.66</td>
</tr>
<tr>
<td>2</td>
<td>1.95</td>
<td>10.26</td>
<td>0.52</td>
<td>38.46</td>
<td>17.82</td>
<td>11.22</td>
<td>2.65</td>
<td>75.47</td>
</tr>
<tr>
<td>3</td>
<td>3.39</td>
<td>8.85</td>
<td>0.67</td>
<td>44.78</td>
<td>27.7</td>
<td>10.83</td>
<td>5.26</td>
<td>57.03</td>
</tr>
<tr>
<td>4</td>
<td>3.68</td>
<td>10.87</td>
<td>0.8</td>
<td>50.00</td>
<td>32.93</td>
<td>12.14</td>
<td>7.87</td>
<td>50.82</td>
</tr>
<tr>
<td>5</td>
<td>4.71</td>
<td>10.62</td>
<td>0.9</td>
<td>55.56</td>
<td>43.86</td>
<td>11.39</td>
<td>9.48</td>
<td>52.74</td>
</tr>
<tr>
<td>6</td>
<td>6.69</td>
<td>8.97</td>
<td>1.03</td>
<td>58.25</td>
<td>49.46</td>
<td>12.13</td>
<td>11.47</td>
<td>52.31</td>
</tr>
<tr>
<td>7</td>
<td>5.94</td>
<td>11.78</td>
<td>1.62</td>
<td>43.21</td>
<td>60.64</td>
<td>11.54</td>
<td>11.73</td>
<td>59.67</td>
</tr>
<tr>
<td>8</td>
<td>6.77</td>
<td>11.82</td>
<td>1.35</td>
<td>59.26</td>
<td>64.48</td>
<td>12.40</td>
<td>11.35</td>
<td>70.48</td>
</tr>
<tr>
<td>9</td>
<td>7.4</td>
<td>12.16</td>
<td>1.7</td>
<td>52.94</td>
<td>72.59</td>
<td>12.39</td>
<td>11.92</td>
<td>75.50</td>
</tr>
<tr>
<td>10</td>
<td>8.18</td>
<td>12.22</td>
<td>2.5</td>
<td>40.00</td>
<td>80.5</td>
<td>12.42</td>
<td>13.39</td>
<td>74.68</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data file size 10 MB</th>
<th>data file size 100 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>rrt 5.131 ms</td>
<td>rrt 0.548 ms</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

is more beneficial to execute them as a single operation if they are requesting data from the same storage site or using the same protocol to access data. Figure 1 presents the performance gain by aggregating the file transfer requests. Table 1 shows the overhead in the connection set-up. The test environment includes host machines from the Louisiana Optical Networking Initiative (LONI) network. Transfer operations are performed using the GridFTP protocol. The average round-trip delay times between test hosts are 5.131 and 0.548 ms. As latency increases, the effect of overhead in connection time increases. We see better throughput results with aggregated requests. The main performance gain comes from decreasing the amount of protocol usage and reducing the number of independent network connections.

We have successfully applied job aggregation in the Stork scheduler such that the total throughput is increased by reducing the number of transfer operations. According to the file size and source/destination pairs, data placement requests are combined and processed as a single transfer job. Information about aggregated requests is stored in a transparent manner. A main job that includes multiple requests is defined virtually and it is used to perform the transfer operation. Therefore, users can query and get status reports individually without knowing...
that their requests are aggregated and being executed with others. Stork performs a simple search in the job queue to determine which requests can be aggregated. In this step, our main criteria are that data placement jobs need to have the same user privileges and should be asking for the same data transfer protocol. We have seen a major increase in total throughput of data transfers, especially with small data files, simply by combining data placement jobs based on their source and destination addresses.

5. Throughput optimization

In data scheduling, the effective use of available network throughput and optimization of data transfer speed is crucial for end-to-end application performance. The throughput optimization in the Stork data scheduler is done by opening parallel streams and setting the optimal parallel stream number specific to each transfer. The intelligent selection of this number is based on a novel mathematical model we have developed that predicts the peak point of the throughput of parallel streams and the corresponding stream number for that value. The throughput of \( n \) streams (\( \text{Th}_n \)) is calculated by the following equation:

\[
\text{Th}_n = \frac{n}{\sqrt{a'n^c + b'}}.
\]  

The unknown variables \( a', b' \) and \( c' \) are calculated based on the throughput samplings of three different parallelism data points. The detailed derivation of this equation and variables as well as the selection strategy of the data points \( (n_1, n_2, n_3) \) are explained in our previous work [41].

The throughput increases as the stream number is increased, it reaches its peak point either when congestion occurs or the end-system capacities are reached. Further increasing the stream number does not increase the throughput but it causes a drop down owing to losses. In figure 2, we present the characteristics of the throughput curve for local area network (LAN) and WAN transfers. In both cases, the throughput has a peak point that indicates the optimal parallel stream number and it starts to drop down for larger numbers of streams. The
prediction model we have developed (dynamic model order by Newton’s iteration) can predict the peak point by using the throughput values of parallelism levels 1, 8 and 16. Our model is much more accurate than previous work—Dinda model [35].

In cases where network optimization is not sufficient and the end-system characteristics play a decisive role in limiting the achievable throughput, making concurrent requests for multiple data transfers can improve the total throughput. The Stork data scheduler also enables the users to set a concurrency level in the server set-up and provides multiple data transfer jobs to be handled concurrently. However, it is up to the user to set this property.

We compare concurrency and parallelism through a set of different test cases to demonstrate the situations where a concurrent transfer can also improve an optimized transfer with parallel streams. The first test case shows us the memory-to-memory transfers with parallel streams and different concurrency levels between a local-area workstation with 100 Mbps network interface and a cluster I/O node in the LONI network with 10 Gbps network interface. The speed of the transfer is bound by the local-area workstation. In figure 3a, increasing the concurrency level does not provide a more significant improvement than increasing the parallel stream values. Thus, the throughput of four parallel streams is almost as good as the throughput of concurrency levels 2 and 4. The reason behind this outcome is that the throughput is bound by the interface and the CPU is fast enough and does not present a bottleneck. Next, we look into the case where we conducted transfers between two clusters with 10 Gbps network interfaces in the LONI network (figure 3b). The I/O nodes have four CPU cores in each. With parallel streams, we were able to achieve a throughput value of 6 Gbps. However, when we increase the concurrency level, we were be able to achieve around 8 Gbps with a combination of a concurrency level of 2 and a parallel stream number of 32. This significant increase is due to the fact that a single CPU reaches its upper limit with a single request but, through concurrency, multiple CPUs are used until the network limit is reached.

The effect of concurrency is much more significant for disk-to-disk transfers where multiple parallel disks are available and managed by parallel file systems. Figure 4 presents the results of disk-to-disk transfers between the I/O nodes of

![Figure 3. Concurrency versus parallelism in memory-to-memory transfers. (a) Dsl-condor–Eric 100 Mbps; (b) Oliver–Poseidon 10 Gbps. (a,b) Red solid line, con-level = 1; green dashed line, con-level = 2; blue dashed line, con-level = 4. (Online version in colour.)](http://rsta.royalsocietypublishing.org/)

*Phil. Trans. R. Soc. A* (2011)
two clusters in the LONI network with 10 Gbps network interfaces. The disk system is managed by the Lustre parallel file system [42]. The x-axis represents the range of parallel stream numbers and the concurrency level is also ranged...
between 1 and 10. As the concurrency level is increased, the size of the data is divided among the concurrent requests totalling to 12 Gbps. While the total throughput presents the total number of bytes transferred in unit time, the average throughput is calculated by the number of bytes transferred in unit time by each concurrent request. Parallel streams improve the throughput for a single concurrency level by increasing it from 500 to 750 Mbps. However, owing to serial disk access, this improvement is limited. Only by increasing the concurrency level can we improve the total throughput. The throughput in figure 4a is increased to 2 Gbps at a concurrency level of 4. After that point, increasing the concurrency level causes the throughput to be unstable with sudden ups and downs in throughput; however, it is always around 2 Gbps. This value is due to the end-system CPU limitations. If we increase the node number better throughput results could be seen. As we look into the figure for average throughput, the transfer speed per request falls as we increase the concurrency level (figure 4b).

We tested our throughput optimization embedded into the Stork data scheduler by combining with it different concurrency levels set for the server. Similar concurrency and data sizes are used as in the previous experiment for disk-to-disk transfers. For each concurrency level, we have submitted 30 jobs to the server and measured the total throughput as the total amount of data transferred divided by the time difference between the first job submitted to the scheduler and the last job finished. The average throughput is calculated as the throughput of each data transfer job separately based on the job start and finish time. When we look into the total throughput results in figure 5a, similar trends are seen. The optimization service provides 750 Mbps throughput for a single concurrency level and it increases up to 2 Gbps for a concurrency level of 4. The average throughput decreases more steeply after that level (figure 5b). Also the optimal parallel stream number decreases and adapts to the concurrency level (figure 5c). From the experiments, it can be seen that an appropriately chosen concurrency level may improve the transfer throughput significantly.

6. Conclusion

In this paper, we have discussed the limitations of the traditional CPU-oriented batch schedulers in handling the challenging data management problem of large-scale distributed e-Science applications. We have elaborated on how we can bring the concept of ‘data awareness’ to several of the most crucial distributed computing components, such as scheduling, workflow management and end-to-end throughput optimization. In this new paradigm, data placement activities are represented as full-featured jobs in the end-to-end workflow, and they are queued, managed, scheduled and optimized via a specialized data-aware scheduler.

As part of this new paradigm, we have developed a set of tools for mitigating the data bottleneck in distributed computing systems, which consists of three main components: a data scheduler that provides capabilities, such as planning, scheduling, resource reservation, job execution and error recovery for data movement tasks; integration of these capabilities with the other layers in

Phil. Trans. R. Soc. A (2011)
distributed computing such as workflow planning; and further optimization of
data movement tasks via aggregation of data transfer jobs considering their
source and destination addresses, and through application-level throughput
optimization. We have described how these last two features are implemented
in Stork and their effects on end-to-end data transfer performance. Our results
show that the optimizations performed by the Stork data scheduler help to
achieve much higher end-to-end throughput in data transfers than non-optimized
approaches.

We believe that the Stork data scheduler and this new ‘data-aware distributed
computing’ paradigm will have an impact on all traditional compute and data-
intensive e-Science disciplines, as well as new emerging computational areas in
the arts, humanities, business and education that need to deal with increasingly
large amounts of data.

This project is in part sponsored by the National Science Foundation under award numbers
CNS-0846052 (CAREER), CNS-0619843 (PetaShare), OCI-0926701 (Stork) and EPS-0701491
(CyberTools), and by the Board of Regents, State of Louisiana, under contract number

References
1 Hey, T. & Trefethen, A. 2003 The data deluge: an e-science perspective. In Grid computing—
2 CCSP. 2003 Strategic plan for the US Climate Change Science Program. A Report by the Climate
Change Science Program and the Subcommittee on Global Change Research. Washington, DC: CCSP.
3 Kosar, T., Balman, M.,Suslu, I.,Yildirim, E. & Yin, D. 2009 Data-aware distributed computing
In Proc. 24th Int. Conf. on Distributed Computing Systems (ICDCS’04), Tokyo, Japan, 23–26,
5 SCOOP. SURA Coastal Ocean Observing and Prediction. See http://scoop.sura.org/.
6 UCoMS. Ubiquitous Computing and Monitoring System for discovery and management of
energy resources. See http://www.ucoms.org/overview.html.
8 WCER. Wisconsin Center for Education Research digital video processing project. See
http://www.wcer.wisc.edu/.
10 MSCFD. Multiscale Computational Fluid Dynamics at LSU. See http://www.cct.lsu.edu/
IGERT/.
11 DOE. 2004 Report from the DOE Office of Science Data-Management Workshops, San Jose, CA,
Challenge.
12 Tierney, B. L., Lee, J.,Crowley, B., Holding, M., Hylton, J. & Drake, F. L. 1999 A network-
aware distributed storage cache for data-intensive environments. In Proc. 8th Int. Symp. on
High Performance Distributed Computing (HPDC’99), Redondo Beach, CA, 3–6 August 1999,
Computing and data grids for science and engineering. Dallas, TX: Supercomputing.
14 Allcock, B. et al. 2001 Secure, efficient data transport and replica management for high-
15 Allcock, B. et al. 2002 Data management and transfer in high performance computational grid
20 ROOT. 2006 An object oriented data analysis framework. See http://root.cern.ch
26 iRODS. The Integrated Rule Oriented Data System. See http://www.irods.org/.


