Stork Data Scheduler: Mitigating the Data Bottleneck in e-Science

By Tevfik Kosar[1,3]†, Mehmet Balman[2], Esma Yildirim[1], Sivakumar Kulasekaran[3], and Brandon Ross[3]

[1]Department of Computer Science & Engineering, State University of NewYork, Buffalo, NY, USA
[2]Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[3]Center for Computation & 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 I/O 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, I/O scheduling, throughput optimization, e-Science, Stork.

1. Introduction

Scientific applications generate increasingly large amounts of data, often referred as the "data deluge" (Hey & Trefethen 2003), 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 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.

According to the 'Strategic Plan for the US Climate Change Science Program (CCSP)', one of the main objectives of the future research programs should be *"Enhancing the data management infrastructure"*, since *"The users should be able*

† Corresponding author: tkosar@buffalo.edu

to focus their attention on the information content of the data, rather than how to discover, access, and use it." (CCSP 2003). This statement by CCSP summarizes the goal of many cyberinfrastructure efforts initiated by NSF, DOE and other federal agencies, as well the research direction of several leading academic institutions. This is also the main motivation for our 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) is either embedded in the computation and causes the computation to delay, or is performed by simple scripts which 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** (Kosar *et al.* 2009).

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 (Kosar & Livny 2004). 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 modeling (SCOOP); oil flow and reservoir uncertainty analysis (UCoMS); numerical relativity and black hole collisions (NumRel); educational video processing and behavioral assessment (WCER); digital sky imaging (DPOSS); and multiscale computational fluid dynamics (MSCFD).

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 data management technology (DOE 2004). 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 (Kosar & Livny 2004). 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."*

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 Section 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 (Section 4), and the application-level throughput estimation and optimization service (Section 5). We conclude the paper in Section 6.

2. Background

Several previous studies address data management for large-scale applications (Tierney *et al.* 1999; Johnston *et al.* 2000; Allcock *et al.* 2001a,b,c; Ranganathan *et al.* 2002, 2004; Venugopal *et al.* 2004; ROOT 2006). 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 Service (RFT) (Ravi *et al.* 2002), the Lightweight Data Replicator (LDR) (Koranda & Moe 2007), and the Data Replication Service (DRS) (Chervenak *et al.* 2005) 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 GridFTP (Allcock *et al.* 2001a,b) which is a secure and reliable data transfer protocol especially developed for high-bandwidth wide-area networks.

Beck *et al.* (1999) introduced Logistical Networking which 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), iRODS, and the Storage Resource Manager (SRM) were developed to provide a uniform interface for connecting to heterogeneous data resources and accessing replicated data sets.

Thain *et al.* (2004) introduced the Batch-Aware Distributed File System (BAD-FS), which was followed by a modified data-driven batch scheduling system (Bent 2005). 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 AFS as well as over traditional CPU-centric batch scheduling techniques which are using remote I/O.

According to Stockinger (2005a,b), 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.* 2001) where theoretical models for data-intensive job scheduling are discussed. In that work, a cost model is created that can determine if 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.* 2005b). In that work, they analyze 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 (Crowcroft *et al.* 1998; Hacker *et al.* 2002; Lu *et al.* 2005; Altman *et al.* 2006). They all have specific constraints and assumptions. Also the correctness of the model is proved

with simulation results mostly. Hacker *et al.* (2002) claim that the total number of streams behaves like one giant stream that transfers in capacity of total of each streams' achievable throughput. However, this model only works for uncongested networks. Thus it is not be able to predict when the network will be congested. Another study (Crowcroft *et al.* 1998) declares the same theory but develops a protocol which at the same time provides fairness. Lu *et al.* (2005) models the bandwidth of multiple streams as a partial second order polynomial equation and needs two different throughput measurement of different stream numbers to predict the others. In another model, the total throughput always shows the same characteristics (Altman *et al.* 2006) depending on the capacity of the connection as the number of streams increases and 3 streams are sufficient to get a 90% utilization. None of the existing studies are able to accurately predict optimal number of parallel streams for best data throughput in a congested network.

3. Stork Data Scheduler

The Stork data scheduler (Kosar 2004; 2005a, b) 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 FTP, 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 multiscale computational fluid dynamics resulting in breakthrough research (Kola *et al.* 2004; Kosar *et al.* 2005b; Ceyhan *et al.* 2008; SCOOP; UCoMS; DPOSS; WCER; NumRel; MSCFD).

Using Stork, the users can transfer very large data sets via a single command. The checkpointing, error recovery and retry mechanisms ensure the completion of the tasks even in case of unexpected failures. Multi-protocol support makes Stork a very powerful data transfer tool. This feature does not only allow Stork to access and manage different data storage systems, but can also 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 to 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 complimentary, 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 lifecycle 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 due 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 which 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 workflow planning tool Pegasus and workflow execution tool Condor DAGMan. This preliminary implementation was used in coastal hazard prediction (SCOOP) and reservoir uncertainty analysis (UCoMS) applications. We have developed a preliminary model to choose between remote-I/O versus 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:

$$Nr - Ns < Wl + Rl \tag{3.1}$$

where Rl is the time to read from local disk to local memory, Wl is the time to write from local memory to local disk, Ns is the time to send data over the network via a staging protocol, and Nr 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 network, or your local disk performance is slow, remote-I/O might be advantageous over staging. Otherwise, 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 traveled. 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.

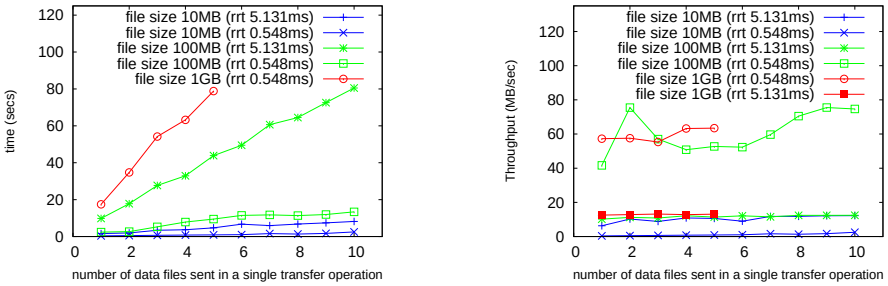

Figure 1. The Effect of Connection Time in Data Transfers

4. Request Aggregation

We have implemented aggregation of data transfer requests in Stork data scheduler in order to minimize the overhead of connection setup 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 overall performance especially for transfers of data sets with small file sizes.

The Stork data scheduler accepts multiple data transfer jobs in a nondeterministic order. For each transfer operation, we need to initialize the network transfer protocol and setup a connection to a remote data transfer service. A single transfer t consists of t_{setup} in which we initialize the transfer module and prepare a connection to transfer the data, and $t_{transfer}$ in which we transfer data using the data transfer protocol over a network. Although this connection time is small compared to the total duration spent for transferring the data, it becomes quite important if there are hundreds of jobs to be scheduled. We target on minimizing the load put by t_{setup} by multiple transfer jobs. Instead of having separate setup operations such as $t_1 = t_{setup} + t_{1transfer}$, $t_2 = t_{setup} + t_{2transfer}$, ..., $t_n = t_{setup} + t_{ntransfer}$, we aggregate multiple requests to improve the total transfer time $t = t_{setup} + t_{1transfer} + t_{2transfer} + ... + t_{ntransfer}$.

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 data scheduler needs to execute separately. Instead of initiating each request one at a time, it 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 is given to show the overhead in the connection setup. The test environment includes host machines from the LONI network. Transfer operations are performed using the GridFTP protocol. The average Round-trip delay times between test hosts are 5.131ms and 0.548ms. 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.

Table 1. *Request Aggregation and the Overhead of Connection Time in Data Transfers*

f_{num}	time (secs)	Thrpt (MBps)	time (secs)	Thrpt (MBps)	time (secs)	Thrpt (MBps)	time (secs)	Thrpt (MBps)
1	1.59	6.29	0.39	25.64	9.81	10.19	2.4	41.66
2	1.95	10.26	0.52	38.46	17.82	11.22	2.65	75.47
3	3.39	8.85	0.67	44.78	27.7	10.83	5.26	57.03
4	3.68	10.87	0.8	50.00	32.93	12.14	7.87	50.82
5	4.71	10.62	0.9	55.56	43.86	11.39	9.48	52.74
6	6.69	8.97	1.03	58.25	49.46	12.13	11.47	52.31
7	5.94	11.78	1.62	43.21	60.64	11.54	11.73	59.67
8	6.77	11.82	1.35	59.26	64.48	12.40	11.35	70.48
9	7.4	12.16	1.7	52.94	72.59	12.39	11.92	75.50
10	8.18	12.22	2.5	40.00	80.5	12.42	13.39	74.68
	RRT 5.131ms		RRT 0.548ms		RRT 5.131ms		RRT 0.548ms	
	data file size 10MB				data file size 100MB			

f_{num}	time (secs)	Thrpt (MBps)	time (secs)	Thrpt (MBps)
1	79.74	12.54	17.46	57.27
2	155.78	12.83	34.74	57.57
3	228	13.15	54.2	55.35
4	312.36	12.80	63.25	63.24
5	381.86	13.09	78.82	63.44
	RRT 5.131ms		RRT 0.548ms	
	data file size 1GB			

f_{num}: number of data files sent in a single transfer operation

We have successfully applied job aggregation in Stork scheduler such that total throughput is increased by reducing the number of transfer operations. According to the file size and source/destination pairs, data placement request 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 figure out which requests can be aggregated. In this step, our main criteria is that data placement jobs need to have the same user privileges and should be asking for the same data transfer protocol. We have seen 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, 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

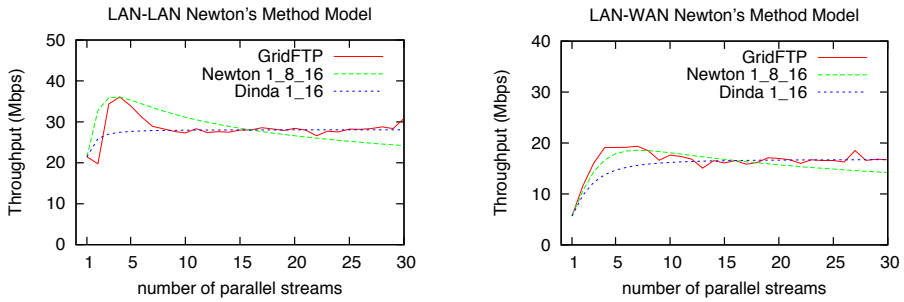

Figure 2. Model Application for Parallel Stream Optimization

and the corresponding stream number for that value. The throughput of n streams (Th_n) is calculated by the following equation:

$$Th_n = \frac{n}{\sqrt{a'n^{c'} + b'}} \tag{5.1}$$

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 selection strategy of data points (n_1, n_2, n_3) are explained in our previous work (Yildirim *et al.* 2010).

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 due to losses. In Figure 2, we present the characteristics of the throughput curve for local-area and wide-area transfers. In both cases, the throughput has a peak point which 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 compared with the previous work –Dinda Model (Lu *et al.* 2005).

In cases where network optimization is not sufficient and the end-system characteristics play a decising 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 server setup 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 3.a, increasing the concurrency level does not provide a more significant improvement than increasing the parallel stream values. Thus, the thrughput of 4 parallel streams is almost as good as the throughput of concurrency levels 2 and 4. The reason behind this outcome is that

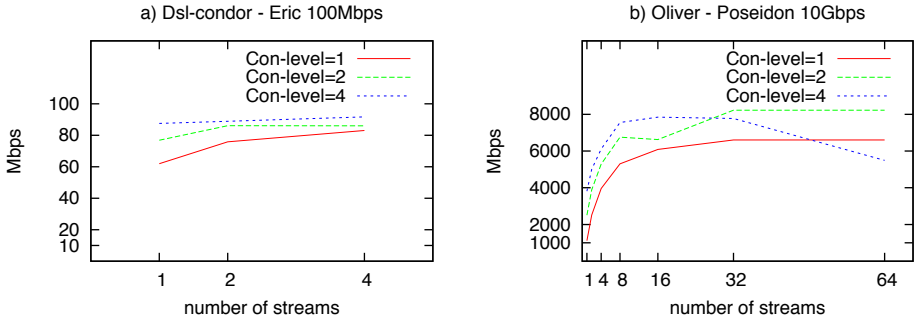

Figure 3. Concurrency vs. Parallelism in Memory-to-memory Transfers

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 3.b). The I/O nodes have 4 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 concurrency level of 2 and 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 utilized 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 two clusters in the LONI network with 10 Gbps network interfaces. The disk system is managed by Lustre parallel file system (Lustre). 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 totaling to 12 Gbps. While the total throughput presents the total number of bytes transferred in unit time, the average throughput is calculated by the amount of bytes transferred in unit time by each concurrent request. Parallel streams improve the throughput for single concurrency level by increasing it from 500 Mbps to 750 Mbps. However, due to serial disk access this improvement is limited. Only by increasing the concurrency level can we improve the total throughput. The throughput in Figure 4.a is increased to 2 Gbps at concurrency level 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 4.b).

We tested our throughput optimization embedded into the Stork data scheduler combining with 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

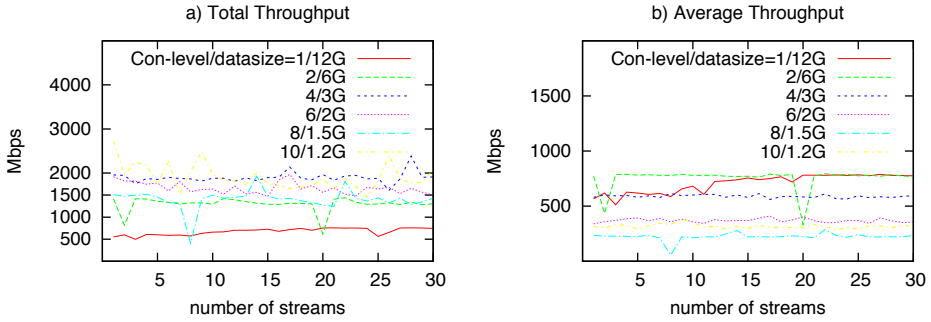

Figure 4. Concurrency vs. Parallelism in Disk-to-disk Transfers

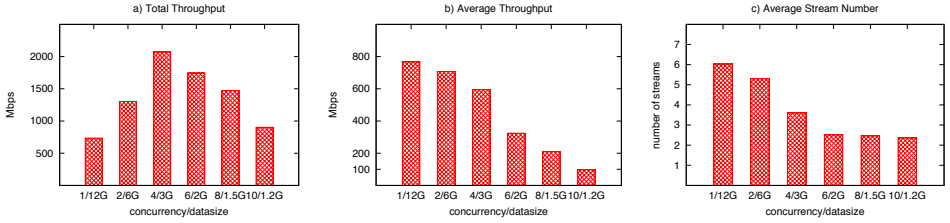

Figure 5. Concurrency vs Parallel Stream Optimization in the Stork Data Scheduler

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 5.a, similar trends are seen. The optimization service provides 750 Mbps throughput for single concurrency level and it increases upto 2 Gbps for concurrency level 4. The average throughput decreases more steeply after that level (Figure 5.b). Also the optimal parallel stream number decreases and adapts to the concurrency level (Figure 5.c). 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 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 which provides capabilities such as planning, schedul-

ing, resource reservation, job execution, and error recovery for data movement tasks; integration of these capabilities with the other layers in 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 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 compared to non-optimized approaches.

We believe that Stork data scheduler and this new 'data-aware distributed computing' paradigm will impact all traditionally compute and data-intensive e-Science disciplines, as well as new emerging computational areas in the arts, humanities, business and education which 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 NSF/LEQSF (2007-10)-CyberRII-01.

References

Allcock, B., Bester, J., Bresnahan, J., Chervenak, A., Foster, I., Kesselman, C., Meder, S., Nefedove, V., Quesnel, D., & Tuecke, S. 2001 Secure, efficient data transport and replica management for high-performance data-intensive computing. In *Proceedings of IEEE Mass Storage Conference, April 2001.*

Allcock, B., Bester, J., Bresnahan, J., Chervenak, A., Foster, I., Kesselman, C., Meder, S., Nefedove, V., Quesnel, D., & Tuecke, S. 2001 Data Management and Transfer in HighPerformance Computational Grid Environments. In *Proceedings of Data Management and Transfer in HighPerformance Computational Grid Environments. Parallel Computing, 2001.*

Allcock, B., Foster, I., Nefedova, V., Chervenak, A., Deelman, E., Kesselman, C., Lee, J., Sim, A., Shoshani, A., Drach, B. & Williams, D. 2001 High-performance remote access to climate simulation data: a challenge problem for data grid technologies. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing, 2001.*

Altman, E., Barman, D., Tuffin, T., & Voinovic, M. 2006 Parallel TCP Sockets: Simple Model, Throughput and Validation. In *Proceedings of INFOCOM , April 2006* , pp. 1-12.

Beck, M., Elwasif, W.R., Plank, J. & Moore, T. 1999 The Internet Backplane Protocol: Storage in the Network. In *Proceedings of the 1999 Network Storage Symposium NetStore99, Seattle, WA, USA*

Bent, J. 2005 Data-driven Batch Scheduling. Ph.D thesis, University of Wisconsin-Madison

CCSP. 2003 Strategic Plan for the US Climate Change Science Program. CCSP Report

Ceyhan, E., Allen, G., White, C. & Kosar, T. A Grid-enabled Workow System for Reservoir Uncertainty Analysis. In *Proceedings of Challenges of Large Applications in Distributed Environments (CLADE 2008) Workshop, June 2008.*

Chervenak, A., Schuler, C., Kesselman, C., Koranda, S., & Moe, B. 2005 Wide area data replication for scientific collaborations. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, November 2005.*

Crowcroft, J. & Oechslin, P. 1998 Differentiated End-to-end Internet Services using a weighted proportional Fair Sharing TCP. *ACM SIGCOMM Computer Communication Review.* **28**, 53-69

DPOSS. The Palomar Digital Sky Survey (DPOSS). *http://www.astro.caltech.edu/george/dposs/.*

Hacker, T.J., Noble, B.D., & Atley, D. 2002 The end-to-end Performance effects of Parallel TCP sockets on a lossy wide area network. In *Proceedings of IPDPS '05*, pp 314.

Hey, T. & Trefethen, A. 2003 The data deluge: An e-science perspective. In Grid Computing - Making the Global Infrastructure a Reality, chapter 36, pp. 809–824. Wiley and Sons, 2003.

IRODS. The Integrated Rule Oriented Data System. iRODS*http://www.irods.org/.*

Johnston, W.E., Gannon, D., Nitzberg, B., Tanner, L.A., Thigpen, B., & Woo, A. 2000 *Computing and data grids for science and engineering*, pp. 52. Supercomputing 2000 CDROM. Dallas, USA

Kiehl, J.T. 1998 The National Center for Atmospheric Research Community Climate Model: CCM3. *Journal of Climate.* **11.6**, 1131-1149

Kola, G., Kosar, T. & Livny, M. A Fully Automated Fault-tolerant System for Distributed Video Processing and Off-site Replication. In *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2004*).

Koranda, S & Moe, 2007 M. Lightweight Data Replicator. *http://www.ligo.caltech.edu/docs/G/G030623-00/G030623-00.pdf*

Kosar, T. & Livny, M. 2004 Stork: Making Data Placement a First Class Citizen in the Grid. In *Proc. of ICDCS*, March 2004, pp. 342-349.

Kosar, T. 2005 Data Placement in Widely Distributed Systems. Ph.D thesis, University of Wisconsin-Madison.

Kosar, T., Kola, G., Livny, M., Brunner, R.J. & Remijan, M. Reliable, Automatic Transfer and Processing of Large Scale Astronomy Data Sets. In *Proceedings of Astronomical Data Analysis Software and Systems (ADASS), 2005.*

Kosar, T., Balman, M., Suslu, I., Yildirim, E., & Yin., D. 2009. Data-Aware Distributed Computing with Stork Data Scheduler. In *Proceedings of the SEE-GRID-SCI'09, Istanbul, Turkey, December 2009*

Lu, D., Qiao, Y., Dinda, P.A., & Stamante, F.E. 2005 Modeling and Taming Parallel TCP on the Wide Area Network. In *Proceedings of IPDPS '05* , pp . 682.

LUSTRE Cluster File System, Inc. Lustre: A Scalable, High Performance File System. *http://www.Lustre.org/docs.htm.*

MSCFD. Multiscale Computational Fluid Dynamics at LSU. *http://www.cct.lsu.edu/IGERT/.*

NumRel. Numerical Relativity at LSU. *http://www.cct.lsu.edu/numerical/.*

Ranganathan, R. & Foster, I. 2002 Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002*, pp.352.

Ranganathan, R. & Foster, I. 2004 Computation scheduling and data replication algorithms for data Grids . *Journal of Grid resource management: state of the art and future trends*. 359-373.

Ravi, M. K., Cynthia, H.S., William, E.A. 2002 Reliable File Transfer in Grid Environments. In *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks, 2002*, pp. 737-738.

ROOT. 2006 Object Oriented Data Analysis Framework. *European Organization for Nuclear Research Journal. http://root.cern.ch.*

SCOOP. SURA Coastal Ocean Observing and Prediction. *http://www.ucoms.org/overview.html.*

SRB. The Storage Resource Broker SRB. *http://www.sdsc.edu/srb/.*

SRM. The Storage Resource Managers SRM. *http://sdm.lbl.gov/srm.*

Stockinger, H. 2005 Data Management in Data Grids - Habilitation Overview. Research Lab for Computational Technologies and Applications.

Stockinger, H., Laure, E. & Stockinger, K. 2005. Performance Engineering in Data Grids. *Journal of Concurrency and Computation: Practice and Experience, Wiley Press.***17(2-4)**, 171-191

Stockinger, K., Schikuta, E., Stockinger, H. & Willers, I. 2001 Towards a Cost Model for Distributed and Replicated Data Stores. In *9th Euromicro Workshop on Parallel and Distributed Processing (PDP 2001), IEEE Computer Society Press, Mantova, Italy, February 2001.*

Thain, D., Arpaci Dusseau, A., Bent, J., & Livny, M. 2004 Explicit Control in a Batch Aware Distributed File System. In *Proceedings of the First USENIX/ACM Conference on Networked Systems Design and Implementation, San Francisco, CA, March 2004.*

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 *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing, 1999*, pp. 185-189.

UCoMS. Ubiquitous Computing and Monitoring System for Discovery and Management of Energy Resources. *http://www.scoop.sura.org/*

Venugopal, S., Buyya, R., & Winton, L. 2004 A grid service broker for scheduling distributed data-oriented applications on global grids. In *Proceedings of the 2nd workshop on Middleware for grid computing, Toronto, Canada, 2004*, pp. 75-80.

WCER. Wisconsin Center for Education Research Digital Video Processing Project. *h*ttp://www.wcer.wisc.edu/.

Yildirim, E., Yin, D. & Kosar, T. 2010 Prediction of optimal parallelism level in wide area data transfers. *TPDS 2010*.

Yin, D., Yildirim, E., Sivakumar, K.A., Ross, B. & Kosar, T. 2011 Data Throughput Prediction and Optimization Service for Widely Distributed Many-Task Computing. *IEEE Transactions on Parallel and Distributed Systems-Special Issue on Many-Task Computing (TPDS-SI)*.