Agile Data Streaming for Grid Applications

Wen Zhang[1], Junwei Cao[2,3*], Yisheng Zhong[1,3], Lianchen Liu[1,3], and Cheng Wu[1,3]
[1]*Department of Automation, Tsinghua University, Beijing 100084, China*
[2]*Research Institute of Information Technology, Tsinghua University, Beijing 100084, China*
[3]*Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China*
[*]*Corresponding email: jcao@tsinghua.edu.cn*

Abstract

The grid is proposed to be the new computing infrastructure for cross-domain resource sharing and collaboration in virtual organizations. While most current work in grid computing community focus on adaptability of grid resource management, agility is proposed in this work as another essential requirement, aiming to provide mass customization and personalization on grid computing service provision. This work is focused on data streaming aspects of grid agility. Since volumes of data streams are usually extremely high but available bandwidth and storage are often very limit in a grid environment, an enabling environment for grid data streaming applications is implemented with supports of on-demand data transfers and just-in-time data cleanups in this work. The implementation includes performance sensors, predictors and a scheduler, with real-time measurements, prediction and scheduling capabilities. The GridFTP is utilized and data transfer schedules are performed using the on-the-fly adjustable GridFTP parallelism. Experimental results show that the data streaming environment is agile enough to meet dynamically changing application data processing requirements and scale well regarding to storage and bandwidth usage.

1. Introduction

Agility is considered as an essential feature to many complex system engineering, e.g. business [11], manufacturing [16] and software development [9]. Agility represents flexibility of response to dynamically changing users' requirements. Agility provides mass customization and personalization of services with extreme flexibility without loosing efficiency. This work is focused on agility for grid data streaming applications.

Data management is one of the most challenging issues in grid implementation. While most existing research on data grids prefer to a *bring-program-to-data* approach, grid applications such as astronomical observations, large-scale simulation and sensor networks may require *bring-data-to-program* supports. For example, LIGO (Laser Interferometer Gravitational-wave Observatory) [6,13] is generating 1TB scientific data per day and trying to benefit from processing capabilities provided by the Open Science Grid (OSG) [20]. Since most OSG sites are CPU-rich and storage-constraint with no LIGO data available, data streaming supports are required in order to utilize OSG CPU resources. In a data streaming scenario, data are transferred and processed constantly as if data were always available from local storage. Meanwhile, processed data have to be cleaned up to save space for the subsequently coming data.

In this work, a data streaming environment is implemented with supports of on-demand data transfers and just-in-time data cleanups. The implementation includes several modules, responsible for real-time measurements, performance prediction, and resource scheduling, respectively. The environment is integrated with Condor [19] for job allocation and Globus [14] for data transfers using GridFTP [3]. Data transfer schedules are performed using the on-the-fly adjustable GridFTP parallelism. Experimental results show that agility of our data streaming environment can meet dynamically changing application data processing requirements with good scalability.

Some existing work on data streaming management are derived from database management systems, e.g. STREAM [4], Aurora [7], NiagaraCQ [8], StatStream [22], and Gigascope [10]. Most of data streaming research in context of grid computing [5,15,17,18] are application specific and scheduling issues are not addressed. The following software implementation is most similar to the work described in this paper:

- *Pegasus* [12,21]. Pegasus handles data transfers, job processing and data cleanups in a workflow manner. Data transfers and processing are executed sequentially. In our environment, data streaming is performed simultaneously as jobs are processed. By carefully optimizing data streaming, our environment makes required data available in an on-demand and just-in-time manner.

- *Streamline* [1,2]. Streamline schedules streaming applications on HPC resources using heuristic methods. Streamline is basically still a job scheduler with consideration of data streams; our environment is focused on scheduling data streaming instead of job processing with consideration of system agility.

This paper is organized as follows: Section 2 provides an overall introduction to data streaming for grid applications, which is implemented in details in Section 3, using repertory policies, real-time measurements, performance prediction and scheduling techniques. Experimental results are included in Section 4 and the paper concludes in Section 5.

2. Grid Data Streaming

A detailed introduction to grid requirements for data streaming is provided in this section. Given an example of grid applications, two major data management methods are proposed: on-demand data transfers and just-in-time data cleanups, which are implemented in Section 3.

2.1. Grid requirements

Traditional grid computing applications prefer to a *bring-program-to-data* approach. Since the cost for transferring large mount of data is always high, it is natural that users try to submit their jobs to computers with data already available and only return data analysis results instead of transferring raw data.

Grid applications such as astronomical observations, large-scale simulation and sensor networks may require working in a *bring-data-to-program* manner. There may be scenarios that owners of data, although ready for data sharing, may be reluctant to allow remote users to execute processing programs on sites for the sake of security or lack of computational resources. For example, in the LIGO project, there are not enough computational resources available at observatories though data have to be generated from observatories. In order to utilize CPU cycles available from the OSG, LIGO data have to be streamed to OSG sites for processing.

On the other hand, it is not always the case that transferring large amount of data requires high bandwidth and local storage space. It is not necessarily either that all data are transferred to local at one time before the processing program is actually started. In this case, data can be processed in the form of *streams*, and corresponding implementation requires supports of on-demand data transfers and just-in-time data cleanups described in the following sections.

2.2. On-demand data transfers

It is not always the case that data should be transferred as fast as possible. For data streaming applications, data transfers should be *on-demand* instead of spontaneous. This is to say, data transfers can be controlled to make required data available according to actual data processing speeds. If data arrive too fast and cannot be processed in time, accumulated data may require large amount of storage space over time; if data transfer speed is lower than processing, corresponding jobs may become idle and computational power is not fully utilized.

A data streaming environment has to support multiple applications, each requiring different data types. All applications have to share limit network bandwidth and storage space. For example, most OSG sites are CPU rich but storage limit. Data transfers have to be scheduled in order to share bandwidth save local storage and maximize CPU usage for data processing. The prevailing principle here is *enough is ok*, not the faster transfers the better, nor the more data in storage the better. For data streaming applications, data transfers and processing take place simultaneously. The data streaming environment aims to provide data for applications transparently and seamlessly as if jobs are processing data that were always available from local storage. This can be achieved via careful scheduling especially with constraints of bandwidth and storage.

2.3. Just-in-time data cleanups

Besides data transfer schedules, a cleanup procedure is integrated in the environment to remove obsolete data and save storage space for subsequently coming data. Processed data should be removed from local storage as soon as possible. Just-in-time cleanups are of great importance in data streaming environment since the whole volume of data are usually extremely high. There maybe situations that multiple applications are sharing a same set of data. In this case, data can be only removed after all jobs are processed, otherwise the same set of data have to be transferred for multiple times.

3. The Environment Implementation

As shown in Figure 1, implementation of the data streaming environment is described in this section, including individual processors for performance sensing, prediction, scheduling and execution of data transfers and cleanups.

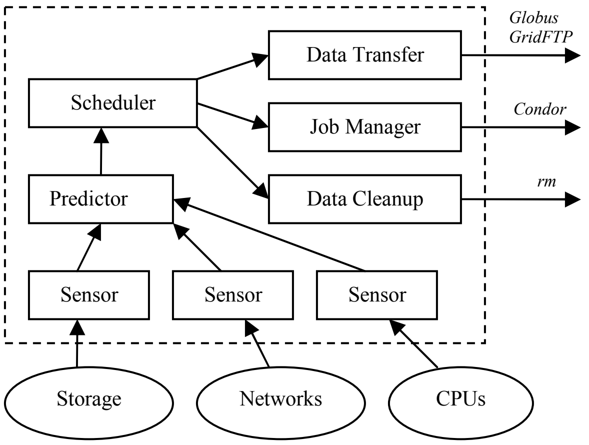

Figure 1. The data streaming environment

- *Sensors*. The grid is a dynamic environment, where performance of resources, including networks, processors and storage, are changing over time. For example, the network is shared by many data transfers and the competition makes available bandwidth for each transfer vary from time to time, so real-time performance information is required. Some sensors are developed to collect such information, which can be used as inputs of the performance predictor.
- *Predictor*. The predictor applies some prediction algorithms to forecast performance for a period of time in the future. Prediction results are inputs of the scheduler.
- *Scheduler*. The scheduler collects performance prediction results (e.g. data transferring and processing speeds) and decides when to transfer data, launch jobs and clean up processed data. Actual executions call external functions provided by Globus and Condor, as shown in Figure 1.

Our optimal goal is to make full use of computational resources, requiring guarantee of data provision, and minimize bandwidth and storage consumptions.

3.1. Real-time measurements

Performance information is of importance to prediction of both data transfers and processing. This can be achieved via real-time performance measurements. Experimental results on GridFTP scalability are included in this section as an example.

We apply GridFTP as the data transfer protocol for cross-domain data replication, parameters of which can be adjusted to get optimal transferring performance, including parallelism, TCP buffer size and buffer size. The parallelism has the most direct impact on data transfer speeds. Our experiments show that the optimal number of data channels is between 8 and 10, as shown in Figure 2. The curve stands for average time of 20 experiments for transferring a data file of 2 GB in seconds, using different parallelisms. It is obvious the data transfer speed increases dramatically with the GridFTP parallelism changing from 1 to 8. When the parallelism reaches over 10, the increment does not result in better performance further.

It is a non-trivial task to determine the proper amount of bandwidth to be allocated for each application running in the Condor pool in terms of utilization and quality of service (QoS) satisfaction. As far as our assignment algorithm concerns, it is transformed to set appropriate GridFTP parallelisms for applications. For convenience, the parallelism is set to 1, 2, 4, 6 or 8 according to data processing speed and status of network. For example, if a certain processing program can consume data of 2 GB in 230 seconds, according to Figure 2, the parallelism can be set to 4 or 6 to guarantee data supply with minimum bandwidth.

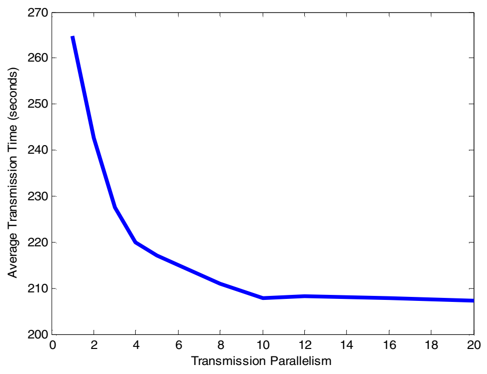

Figure 2. Comparison of data transfer times using different GridFTP parallelisms

Since the grid is a dynamic and resource sharing environment, the same GridFTP parameter may result in different actual data transfer speed over time, and data processing speed in the Condor pool may also vary over time. A well set parallelism may not always match data requirements over a long time, which makes it necessary to evaluate these parameters periodically. In a time interval, if the GridFTP can not supply enough data, its parallelism should be set to its upper neighbors, e.g., from 6 to 8. But we must be cautious to set the parallelism to a lower level unless the redundant transferring speed is observed in several successive intervals. If even the highest parallelism can not meet data processing requirements, the corresponding processor has to be inevitably idle and wait for more available data; if the data transferring speed is high enough, some repertory policy should be applied to avoid data overflow.

3.2. Repertory policies

A repertory strategy with lower and upper limits for each type of data is applied for the scheduler to decide the start and end of data transfers and ensure only reasonable local storage is required. For most OSG sites, the total storage is limit and must be shared by many types of data. On the other hand, it is meaningless to keep processed data. Repertory policies are defined to lower and upper limits of storage space for each type of data.

The lower limit is used to guarantee that data processing can survive network collapse when no data can be streamed from sources to local storage for a certain period of time, which improves system robustness and increases CPU resource utilization. This is also illustrated in Section 4 with detailed experimental results. The upper limit for each application is used to guarantee that the overall amount of data in local storage does not exceed available storage space.

Our strategy can be denoted as $M(l_i, u_i)$, where l_i and u_i stand for lower and upper limits for data type i $(i=1,2,\ldots,n)$ and n is the total number of data types. Lower and upper limits are mainly used as thresholds to control start and end times of data transfers: when data amount scratches the lower limit, more data should be transferred until the amount reaches the upper limit. Since there are also data cleanups involved, data amount keeps changing and varies between lower and upper limits.

3.3. Performance prediction

Data transferring and processing speeds must match each other so as to gain optimized performance with high throughput, so it is necessary to implement real-time monitoring and prediction. Online measurement and prediction of CPU usage is included in this section.

There are many available prediction methods, including nonlinear time-series analysis, wavelet analysis, rough and fuzzy sets. Our implement of fractal prediction for CPU usage can be calculated rapidly with reasonable precision, which can meet our requirement though not necessarily the best algorithm.

The fractal distribution can be described as:

$$N = \frac{C}{r^D},$$

where r is the sample time, an independent variable; N is the usage percent of CPU, a variable corresponding to r; C is a constant to be calculated and D stands for the fractal dimension.

Define a series of initial data N_j $(j=1,\ldots,m)$ and the aggregate sum of i^{th} order can be calculated as:

$$S_{i,j} = \begin{cases} \sum_{k=1}^{j} N_k, i=1 \\ \sum_{k=1}^{j} S_{i-1,k}, i>1 \end{cases}.$$

The fractal dimension $D_{i,j}$ and the constant $C_{i,j}$ can be calculated as:

$$D_{i,j} = \ln\frac{S_{i,j+1}}{S_{i,j}} \Big/ \ln\frac{r_j}{r_{j+1}}$$

$$C_{i,j} = S_{i,j} * r_j^{D_{i,j}},$$

where $r_j=j$ $(j=1,\ldots,m-1)$.

Here the second order aggregated sum is applied since for most cases its prediction result is good enough and its computing overhead is reasonable. In this case the prediction is:

$$N_{m+1} = \frac{S_{2,m} * m^{D_{2,m-1}}}{(m+1)^{D_{2,m-1}}} - S_{2,m} - S_{1,m}$$

Table 1. Fractal prediction of CPU usage (m=15)

N_j	$S_{1,j}$	$S_{2,j}$	$S_{3,j}$	$S_{4,j}$	$D_{1,j}$	$D_{2,j}$	$D_{3,j}$	$D_{4,j}$
39.6	39.6	39.6	39.6	39.6	-1.0641	-1.6280	-2.0324	-2.3479
43.2	82.8	122.4	162	200	-1.2830	-1.8742	-2.3712	-2.7917
56.5	139.3	261.7	424	630	-1.1941	-1.9462	-2.5477	-3.0579
57.1	196.4	458.1	882	1510	-1.1331	-1.9700	-2.6498	-3.2319
56.5	256.9	711.0	1593	3100	-0.9513	-1.9351	-2.6974	-3.3451
47.9	300.8	1011.8	2605	5700	-0.9716	-1.9243	-2.7274	-3.4239
48.6	349.4	1361.2	3966	9670	-0.9035	-1.9046	-2.7445	-3.4805
44.8	394.2	1755.4	5721	15390	-0.7084	-1.8544	-2.7451	-3.5191
34.3	428.5	2183.9	7905	23300	-0.8288	-1.8414	-2.7453	-3.5471
39.1	467.6	2651.5	10557	33850	-0.8653	-1.8385	-2.7469	-3.5689
40.2	507.8	3159.3	13761	47570	-0.8567	-1.8355	-2.7490	-3.5866
39.3	547.1	3706.4	17422	64990	-0.7597	-1.8204	-2.7489	-3.6008
34.3	581.4	4287.8	21710	86700	-0.8260	-1.8171	-2.7492	-3.6126
36.7	618.1	4905.9	26616	113320	-0.8692	-1.8198	-2.7507	-3.6229
38.2	656.3	5562.2	32178	145500	-----	-----	-----	-----
Predicted N_{16}	36.9		Measured N_{16}	38.4		Relative error	3.91%	

One of prediction results is given in Table 1. N_j is an average CPU usage (%) during a 10 seconds' interval. While $D_{1,j}$ varies to some extent, $D_{2,j}$, $D_{3,j}$ and $D_{4,j}$ approach to a stable value respectively. This means that fractal dimensions tend to be fixed and can be applied to make prediction. As shown in Table 1, the relative error of the prediction N_{16} is reasonably good.

A predication of 100 samples is included in Figure 3. While there are some difference between the prediction and measurements, prediction results does follow the trend of CPU usage measurements.

3.4. Scheduling of data streaming

The amount of data in storage varies over time and can be described as the following:

$$\dot{Q}_i(t) = I_i(t) - d_i(t) \quad i=1,2,\cdots,n$$

$$Q_i(0) = 0 \quad i=1,2,\cdots,n,$$

where $\dot{Q}_i(t)$, $I_i(t)$ and $d_i(t)$ stand for the varying rate, assigned transferring bandwidth and processing speed for data type $i(i=1,\ldots,n)$. $\dot{Q}_i(t)$ is the derivative of the data amount in storage, which reflects integrated effects of input and cleanup of data. Data is streamed to local storage on-demand and then processed and removed.

If $I_i(t) = d_i(t)$, there is $\dot{Q}_i(t) = 0$, which means that all data is processed and removed as soon as they are transferred, and theoretically no extra storage is necessary at any time.

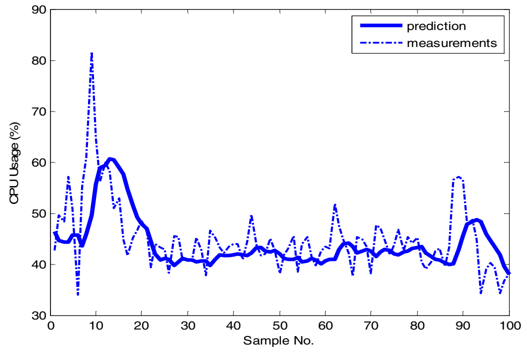

Figure 3. Comparison of 100 prediction and measurements of CPU usage

Data processing programs run constantly to process available data. If no data is locally available, they have to be idle and wait which wastes computational resources. So $d_i(t)$ can be described as:

$$d_i(t) = \begin{cases} 0 & Q_i(t) = 0 \\ >0 & Q_i(t) > 0 \end{cases} \quad i = 1,2,\cdots,n \cdot$$

Sometimes we can just take $d_i(t)$ as a constant, and then we have:

$$d_i(t) = \begin{cases} 0 & Q_i(t) = 0 \\ d_i & Q_i(t) > 0 \end{cases} \quad i = 1,2,\cdots,n \cdot$$

As mentioned before, the total storage available is limit and the constraint is:

$$\sum_{i=1}^{n} Q_i(t) \le S,$$

where S is the total available storage. As introduced in repertory policies, every type of data has its lower and upper limits on storage space, L_i and U_i:

$$L_i \le Q_i(t) \le U_i \quad i = 1,2,\cdots,n$$

It is necessary to set values for lower and upper limits for each data type since these define when to start and end transferring data. In this work these values are set manually, and eventually some heuristic methods can be applied to find more optimal values.

$I_i(t)$ represents data transferring speeds and can be controlled to optimize performance in terms of throughput with the following constraint:

$$\sum_{i=1}^{n} I_i(t) \le I(t),$$

where $I(t)$ is the total available local bandwidth at a given time t. In an actual environment, $I_i(t)$ can be adjustable using carefully selected GridFTP parallelism p for each data type. Possible values of p are 1, 2, 4, 6, and 8 and can be selected as:

$$p = \begin{cases} 1, \tilde{I}_i(t) \le I_i(t,1) \\ 2, I_i(t,1) < \tilde{I}_i(t) \le I_i(t,2) \\ 4, I_i(t,2) < \tilde{I}_i(t) \le I_i(t,4) \\ 6, I_i(t,4) < \tilde{I}_i(t) \le I_i(t,6) \\ 8, \tilde{I}_i(t) > I_i(t,6) \end{cases},$$

where $\tilde{I}_i(t)$ represents the data transferring requirement (in another word, data processing speed) for data type i at a given time t. $I_i(t,p)$ is the estimated data transferring speed using the GridFTP parallelism p for data type i at a given time t. These can be measured and predicted using methods proposed in Sections 3.1 and 3.3.

With above processes and constraints, the optimization goal is to minimize the waste of computational resources, or in another word, to maximize throughput of each data type:

$$\max \int_0^{T_f} d_i(t)dt \quad i = 1,2,\cdots,n,$$

or the whole throughput for all types of data:

$$\max \sum_{i=1}^{n} \int_0^{T_f} d_i(t)dt = \max \int_0^{T_f} \left(\sum_{i=1}^{n} d_i(t) \right) dt,$$

or, if each type of data processing has different privileges or weights:

$$\max \sum_{i=1}^{n} \int_0^{T_f} \omega_i d_i(t)dt = \max \int_0^{T_f} \left(\sum_{i=1}^{n} \omega_i d_i(t) \right) dt$$

where T_f is the evaluation time span and ω_i is the weight of data type $i(i=1,\ldots,n)$.

There could be many other scheduling goals, e.g. fairness of data transferring and processing. Many existing policies in job scheduling can also be applied for scheduling of data streaming. These will be addressed in our future work.

4. Experimental Results

A prototype data streaming environment is implemented to evaluate algorithms proposed in Section 3. A Condor pool is built to manage over 10 workstations in the laboratory, providing computational and storage resources. Data to be processed are available in remote sites rather than machines in the Condor pool, and Globus GridFTP is adopted for data streaming.

4.1. Experiment design

An LIGO data analysis program is used as a case study of data streaming applications. It reads in data from both observatories and calculates statistics for gravitational burst analysis. LIGO data for analysis are transformed from raw data to binary data files in reduced size using an international standard data format.

Data processing programs are submitted to the Condor pool and allocated to available computational nodes according to their requirements. Processing programs run constantly on host machines, and required data are transferred from data sources on remote sites to local storage and fed to corresponding processing programs.

Our data streaming environment with several individual daemons, including sensors, the predictor, the scheduler and data transfer and cleanup agents, is independent from applications. It monitors processing programs, collects information about data transferring and consuming to decide parameters dynamically, i.e. upper and lower limits of repertory policies as thresholds of data transfers, GridFTP parallelisms, and so on. Since the grid is a dynamic environment, scheduling is carried out over and over again and makes new decisions when necessary, e.g. new applications join or data processing speeds and network situations change.

4.2. Performance evaluation - scalability

Three repertory policies are used for performance evaluation of storage scalability, which are illustrated in Figure 4.

- *Continuous streaming without cleanup*. Data transfers are not under any control, i.e. data are transferred constantly, and no cleanup procedure is involved, so data accumulates rapidly, as shown in Figure 4.
- *Continuous streaming with cleanup*. The cleanup process is added. While the required storage space scales better, data still accumulates over time. This is because data transfers are faster than processing. Although processed data can be removed immediately, data waiting to be processed still accumulates and required storage space increases.
- *On-demand streaming with cleanup*. In this scenario, the scheduling algorithm proposed in Section 3 is applied. Since data transfers are controlled using lower and upper limits, data transfers would stop at upper limits and data overflow would not happen.

From experimental results given in Figure 4 we can infer that our data streaming and scheduling mechanism should scale well with applications requiring high volumes of data and long-term runs.

With storage lower limits, our repertory policies also result in tolerance to unexpected situations, e.g. network failures. This guarantees applications can survive network failures which disable data streaming from remote sites. Figure 5 shows such an instance.

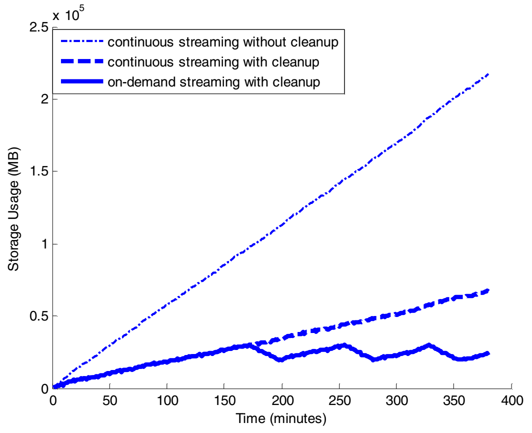

Figure 4. Performance evaluation on storage scalability using 3 repertory strategies

During a many hours' data streaming period, a network broken happens in the middle for a short period of time. While the amount of local available data (represented using data file numbers) drops below the lower limit temporarily, the application is still running and after the network recovers, data storage resumes to a normal level.

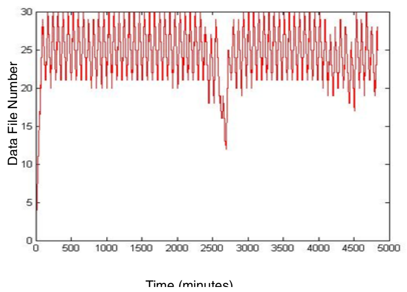

Figure 5. Storage usage tolerant to an unexpected network failure

4.3. Performance evaluation - agility

We investigate agility of our data streaming implementation by setting up an experiment with dynamically changing data processing speeds. This could happen especially when multiple processes are sharing CPU resources. Four typical scenarios are designed in the experiment and a scheduling scheme of data streaming for 4 processing programs is illustrated in Figure 6.

- *data1*. Since the *data1* application has a very high data processing speed, the GridFTP parallelism is set to the highest 8 to meet the requirement. Even data has been streamed as fast as possible, it is obvious that data processing still has to be idle and wait for available data occasionally.

- *data2*. As shown in Figure 6, the data processing speed of data2 application decreases over time, leading to a dynamically changing scheduling scheme for data transferring. The GridFTP parallelism starts from 8 and finally decreases to 2 since processing speed becomes low after 14 minutes.

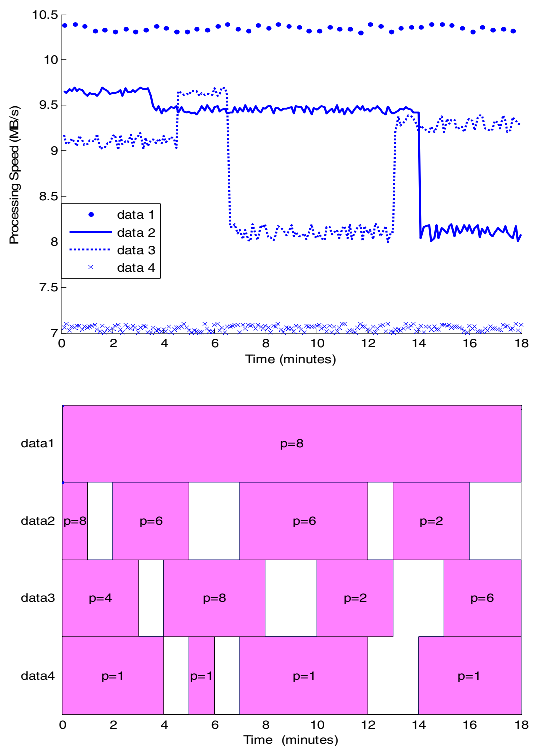

Figure 6. Performance evaluation on agility of data streaming to dynamically changing data processing speeds

- *data3*. The data processing speed of *data3* application changes dramatically over time. The data streaming environment has to adjust the GridFTP parallelism *p* on-the-fly between 2 and 8.
- *data4*. In contrast with *data1* application, *data4* is slow so *p=1* can always satisfy the processing requirement, even with some intervals when no data transfers happen.

In all of above scenarios except *data1*, data transfers start and stop automatically, meeting application requirements as well as avoiding data overflows.

In this work, we provide an integrated grid resource scheduling scheme for data streaming applications with combination of considerations of relevant resources, e.g. CPUs, storage and bandwidth. This improves the system agility to serve individual data processes in a customized and personalized way, though application requirements are dynamically changing.

5. Conclusions

This work is focused on agility aspects of data streaming for grid applications. On demand data transfers and just-in-time data cleanups are proposed to implement system agility and meet application requirements with consideration of scalability. In order to utilize computational resources with limit storage and bandwidth, large amount of data have to be streamed for processing.

A prototype of a data streaming environment is implemented, together with Condor for CPU allocation and Globus GridFTP for data transfers. Some repertory policies are defined, performance measurement and prediction methods are proposed, and scheduling algorithms are implemented. Experimental results show good scalability and agility of our environment.

Ongoing work includes the consideration of data sharing scenarios among multiple data processing applications. Also some heuristic scheduling algorithm is under development for refined performance optimization. We are also investigating the method to combine data streaming scheduling with traditional job scheduling so that system agility could be improved further.

Acknowledgement

This work is supported by Ministry of Education of China under the 211/15 project "National University Instrument and Resource Sharing Systems", and Ministry of Science and Technology of China under the national 863 high-tech R&D program (grants No. 2006AA10Z237, No. 2007AA01Z179 and No. 2008AA01Z118).

Junwei Cao would like to express his gratitude to Prof. Erik Katsavounidis of LIGO (Laser Interferometer Gravitational-wave Observatory) Laboratory at Massachusetts Institute of Technology for their long-term collaboration supports.

References

[1]. B. Agarwalla, N. Ahmed, D. Hilley, and U. Ramachandran, "Streamline: a Scheduling Heuristic for Streaming Applications on the Grid", in *Proc. SPIE Multimedia Computing and Networking*, Vol. 6071, 2006.

[2]. B. Agarwalla, N. Ahmed, D. Hilley, and U. Ramachandran, "Streamline: Scheduling Streaming Applications in a Wide Area Environment", *Multimedia Systems*, Vol. 13, No. 1, pp. 69-85, 2007.

[3]. B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and S. Tuecke, "Data Management and

Transfer in High Performance Computational Grid Environments", *Parallel Computing*, Vol. 28, No. 5, pp. 749-771, 2002.

[4]. A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom, "STREAM: The Stanford Stream Data Manager", *IEEE Data Eng. Bull.*, Vol. 26, No. 1, pp.19-26, 2003.

[5]. V. Bhat, S. Klasky, S. Atchley, M. Beck, D. McCune, and M. Parashar, "High Performance Threaded Data Streaming for Large Scale Simulations", in *Proc. 5th IEEE/ACM Int. Workshop on Grid Computing*, pp. 243-250, 2004.

[6]. D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, "A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis", in I. J. Taylor, D. Gannon, E. Deelman, and M. S. Shields (Eds.), *Workflows for eScience: Scientific Workflows for Grids*, Springer Verlag, 2006.

[7]. D. Carney, U. Cetinterne, M. Cherniack, et. al., "Monitoring Streams: A New Class of Data Management Applications", in *Proc. Int. Conf. on Very Large Data Bases*, pp. 215-225, 2002.

[8]. J. Chen, D. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases", in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 379-390, 2000.

[9]. A. Cockburn, *Agile Software Development*, Addison-Wesley, 2001.

[10]. C. Cranoe, T. Johnson, V. Shkapenyuk, and O. Spatscheck, "Gigascope: a Stream Database for Network Applications", in Proc. *ACM SIGMOD Int. Conf. on Management of Data*, pp. 379-390, 2003.

[11]. S. Davis, and B. J. Pine II, *Mass Customization: The New Frontier in Business Competition*, Harvard Business School Press, 1999.

[12]. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, et. al., "Mapping Abstract Complex Workflows onto Grid Environments", *J. Grid Computing*, Vol. 1, No. 1, pp. 25-39, 2003.

[13]. E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, et. al., "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists", in *Proc. 11th IEEE Int. Symp. on High Performance Distributed Computing*, pp. 225-234, 2002.

[14]. I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *Int. J. Supercomputer Applications*, Vol. 11, No. 2, pp. 115-128, 1997.

[15]. G. Fox, H. Gadgil, S. Pallickara, M. Pierce, R. L. Grossman, et. al., "High Performance Data Streaming in Service Architecture", *Technical Report, Indiana University and University of Illinois at Chicago*, July 2004

[16]. P. T. Kidd, *Agile Manufacturing: Forging New Frontiers*, Addison-Wesley, 1994.

[17]. S. Klasky, S. Ethier, Z. Lin, K. Martins, D. McCune and R. Samtaney, "Grid-Based Parallel Data Streaming Implemented for the Gyrokinetic Toroidal Code", in *Proc. ACM/IEEE Supercomputing Conf.*, 2003.

[18]. R. Kuntschke, T. Scholl, S. Huber, A. Kemper, A. Reiser, et. al, "Grid-Based Data Stream Processing in e-Science", in *Proc. 2nd IEEE Int. Conf. on e-Science and Grid Computing*, 2006.

[19]. M. Litzkow, M. Livny, and Matt Mutka, "Condor – A Hunter of Idle Workstations", in *Proc. of 8th Int. Conf. on Distributed Computing Systems*, pp. 104-111, 1988.

[20]. R. Pordes for the Open Science Grid Consortium, "The Open Science Grid", in *Proc. Computing in High Energy and Nuclear Physics Conf.*, Interlaken, Switzerland, 2004.

[21]. A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi, "Scheduling Data-intensive Workflow onto Storage-Constrained Distributed Resources", in *Proc. 7th IEEE Int. Symp. on Cluster Computing and the Grid*, Rio de Janeiro, Brazil, pp. 401-409, 2007.

[22]. Y. Zhu and D. Shasha, "Statstream: Statistical Monitoring of Thousands of Data Streams in Real Time", *Technical Report TR2002-827, CS Dept, New York University*, 2002.