Concurrent and Storage-aware Data Streaming for Data Processing Workflows in Grid Environments

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

Data streaming applications, usually composed with sequential/parallel data processing tasks organized in a workflow manner, bring new challenges to workflow scheduling and resource allocation in grid environments. Due to high volumes of data and relatively limit storage capability, resource allocation and data streaming have to be storage aware. Also in order to improve system performance, data streaming and processing have to be carried out concurrently. In this paper, Genetic Algorithm is adopted for workflow scheduling, based on on-line measurement and fractal prediction. On-demand data streaming is introduced to avoid data overflow using repertory strategies. Experimental results show that balance among task executions with on-demand data streaming is required to improve overall performance, avoid system bottlenecks and backlogs of intermediate data, and increase data throughput of data processing workflows as a whole.

1. Introduction

Grid computing [1] enables cross-domain resource sharing of CPU cycles, data storage and even scientific instruments. Some grid resources, e.g. astronomical observatories, large simulations and sensor networks, are generating large amount of data every day. It brings more challenges to process these data streams.

While most existing research on data grids prefer to a *bring-program-to-data* approach, data streaming applications require *bring-data-to-program* supports. For example, modern physical experiments, such as LIGO (Laser Interferometer Gravitational-wave Observatory) [2], may produce terabytes of scientific data per day without enough data processing capability onsite. In order to make use of CPU cycles provided by some open grid infrastructures, e.g. the Open Science Grid [3], data have to be streamed constantly to remote grid sites for processing.

In order to make good use of all grid resources, e.g. CPU cycles, storage and network bandwidth, data streaming and processing have to be *cooperative*，i.e., they should be scheduled to match each other as much as possible to gain high processing efficiency with just reasonable resources. For example, if data arrive faster than the processing speed, accumulated data will require more storage; if data arrival is slower than the processing speed, CPUs have to be idle and waiting for available data. What's more, data supply and processing should be carried out in a concurrent way, i.e., data supply and processing are running simultaneously and constantly, rather than subsequently, to make full utilization of resources. A scheduler is required to coordinate these separate processes to make better use of all these grid resources. Due to the dynamism of resources performance, such scheduling scheme should be made periodically with updated information based on measurement and prediction.

Since most of open grid sites may be CPU-rich and storage-limit with no data available, storage availability has to be considered during task scheduling and resource allocation. Data streaming have to be *storage-aware*, which means data should arrive on-demand instead of the-faster-the-better. Meanwhile, processed data have to be cleaned up to save storage space for the subsequently coming data.

In this work, a grid data streaming application is decomposed into several tasks that interact with each other. Tasks are executed in a manner of workflows, in which preceding tasks' outputs are used as inputs of subsequent ones. Task execution times on different computational nodes vary as CPUs, memories, software and workloads are different. In the computing pool where the workflow is deployed and executed, data from the remote source and the medium products of each stage are stored in a network file system (NSF), which guarantees that the communication time, or more exactly, transfer time for medium items to its direct children can be neglected. So only the execution

times for tasks (or in another name, stages) here are counted. Genetic Algorithm (GA) [4] is adopted for workflow scheduling, based on on-line performance measurement and fractal prediction. Due to extremely high volumes of data to be streamed and processed and relatively shortage of available storage, on-demand data streaming is introduced to avoid data overflow using repertory strategies [reference], and implemented using GridFTP [5] of the Globus Toolkit [6][7]by tuning the data transfer parallelism [more reference]. Experimental results included in this work show that our approach makes better use of CPU cycles as well as improving data throughput of overall workflows with storage constraints.

This paper is organized as follows: Section 2 provides a formal description of grid data streaming issues; system implementation is described in Section 3 with algorithms for online measurement and fractal prediction, the genetic algorithm for task scheduling, and repertory strategies for on-demand data streaming; experimental results are included in Section 4. A summary of technical discussion and related work can be found in Section 5 and Section 6 respectively; Section 7 concludes the paper with a brief introduction to future work.

2. Problem Statement

Computational resources and bandwidth should be allocated in an integrated and cooperative way, to gain high throughput and small backlogs (both of these terms will be defined later) with just reasonable resources, just as revealed in our previous work [8].

A data streaming workflow is usually composed with sequential/parallel tasks. Grid resources, including computational and storage resources, are allocated to tasks so as to meet the quality of service (QoS) requirement of the overall application.

A workflow can be represented using a coarse-grain directed acyclic graph (DAG), denoted as $G=(V,E)$, where, V is the set of tasks and E is the set of edges, with a data stream input. Each node s_i of a graph represents a continuously running application task with directions of data flows denoted by edges. Each edge $(i,j) \in E$ represents a direction of a data flow such that task s_j waits for data to arrive from task s_i before execution, namely, task s_i is the immediate parent of task s_j. The goal of our resource allocation is to maximize data throughput of the whole workflow, while being aware of storage requirements.

Define two sets S and R, denoting the set of tasks and resources respectively:

$$s_i \in S, i = 1,2,\cdots,m$$

$$r_j \in R,\ j = 1,2,\cdots,n,$$

where m and n are tasks and candidate resource numbers, respectively. We are trying to find a mapping from S to R:

$$f : s_i \to r_j,\ s_i \in S,\ r_j \in R,\ i = 1,2,\cdots,m,\ j = 1,2,\cdots,n$$

from the n^m possible mapping schemes to get the optimal performance in terms of data streaming throughput. The searching scope can be diminished if constraints of application QoS and eligible resources for s_i are considered:

$$R_i = \{r_k \mid r_k \text{ is eligible for } s_i, r_k \in R\},\ i = 1,2,\cdots,m$$

$$R = \bigcup_{i=1}^{m} R_i .$$

Let N_i stand for the number of elements in R_i. The number of possible mapping schemes is:

$$N = \prod_{i=1}^{m} N_i .$$

Then our task is to select the optimal or at least satisfying one from the N possible choices. Data streaming also plays a key role in our scenario, and our approach is to tune the data transfer parallelism p on-the-fly in the GridFTP tool to get appropriate speed of data transfer for corresponding tasks to optimize performance with maximum data throughput. Other parameters for data streaming, such as the TCP buffer size, also influent data provision, but so far these are not taken into account.

In a scheduling scheme, workflow deployment is fixed, i.e., tasks are assigned to certain computing resources and they will not be migrated to others. But the data supply speed is adjusted periodically according to processing speed and storage usage, by adjusting parallelism of GridFTP. To match data processing and supply, performance prediction of both of them is needed and fractal prediction is applied to give the clues of them in the coming scheduling period with information of past. Tasks are monitored to get information on corresponding data requirements, and at the same time, on-line measurement and prediction of data transfer speed with different parallelisms are carried out. Repertory strategies are applied for on-demand data streaming, where upper and lower limits of repertory are used to stop and resume data transfers, to guarantee data provision with reasonable size of storage. The system architecture and detailed description of algorithm implementation will be described in the following sections.

Just as mentioned above, primitive data items from remote data sources and medium data items produced by tasks in the workflow are stored in the network file system, which means that medium data items can be

transferred to its destinations in a negligible time span. So, when deploy a workflow, only execution times of tasks will be taken into account.

In our scenario, data are transferred and processed in the form of tuples, or more exactly, in successive small files which are even in size. For example, each input file of this workflow contains data acquired in 16 seconds, and each medium data item also exists in form of small files. We esteem the small files as the least unit of data, and definition of throughput and backlogs are made in this metric.

Throughput is defined as the number of small files (transferred from their remote sources) processed in the given evaluation time span, while backlog of each task means the number of data files (as their input from their direct parents) accumulated in its stage which are not processed in time. Obviously, high throughput with low backlogs is desirable, which requires high utilization of computing and bandwidth resources and balance among tasks in the workflow, as we will see in the evaluation part of this paper.

3. System Implementation

This part elaborates on the system architecture and some key algorithms, which will be evaluated in the next section.

3.1 System Architecture

Our system for grid data streaming workflows is based on Globus for grid data transfers using GridFTP and Condor [9] for local task management, as shown in Figure 1.

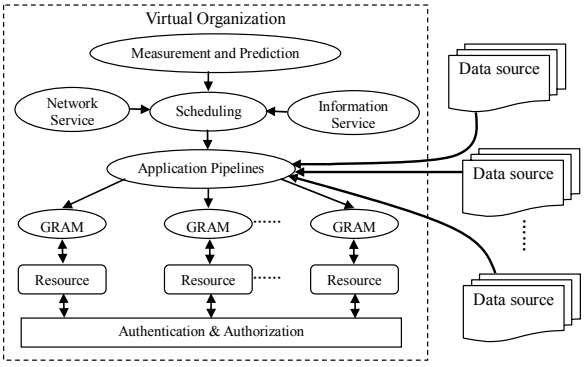

Figure 1. System Architecture

Through authentication and authorization (e.g. Grid Security Infrastructure), grid resources, including CPU cycles, storage, networks and so on, form a virtual organization to share resources and collaborate to provide support for data streaming applications, whereas data sources are at remote sites and data are streamed to application workflows to be processed and cleaned up subsequently. Information and network services are used to provide static and dynamic information about grid resources, such as hardware configurations, CPU workloads and network bandwidth in real time. GRAM (Grid Resource Allocation and Management) is responsible for task management, which is integrated with Condor in our case. The details of these modules are out of the scope of this paper. We concentrate on performance measurement and prediction, resource allocation for tasks in workflows and on-demand tuning of data transfers, with detailed information below.

As mentioned in Section 2, our goal is to maximize data throughput of the whole application workflow by allocating appropriate resources to tasks with constraints of available storage. In our resource allocation and scheduling, we also take into account static information of available resources, dynamic information of available processing capabilities in the target environment. We execute resource filtering to select available resources based on application and resource specific policies, to reduce the scope of candidate resources and allocation complexity. Dynamic performance information of resources is mainly used to make an updated allocation scheme over time.

3.2 Measurement and Prediction

Both of data requirement (data processing speed) and data supply speed should be predicted to make as good a match as possible. Tasks in a workflow are deployed to certain computing resources according to the average execution times over a long time, which forms an execution time matrix, e.g. T in subsection 4.1. Tasks are executed repeatedly on successive data files, so such an average execution time matrix makes sense in the long run. But as long as a short scheduling period is concerned, both execution times of tasks and data supply with different parallelism are varying. It is desirable to make prediction of such performance from measured historical values for the next scheduling period, to carry out just on-demand data transfers. As such scheduling is performed periodically, performance measurement and prediction will be made repeatedly. Obviously, there is a tradeoff between precision and calculation speed of such measurement and prediction.

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,…,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,…,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_i	$S_{1,i}$	$S_{2,i}$	$S_{3,i}$	$S_{4,i}$	$D_{1,i}$	$D_{2,i}$	$D_{3,i}$	$D_{4,i}$
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 2. While there are some difference between the prediction and measurements, prediction results does follow the trend of CPU usage measurements.

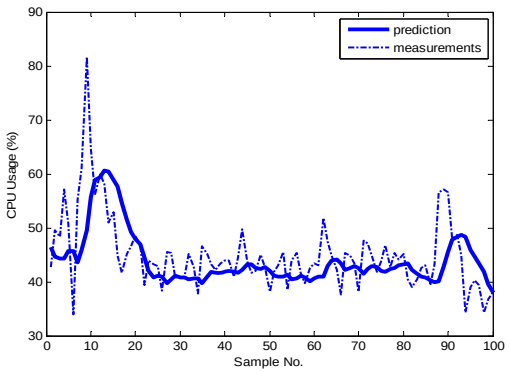

Figure 2. Comparison of 100 prediction and measurements of CPU usage

With some initial performance data in a relatively small amount, we can make a prediction of future performance, such as the execution time of a task on certain resource and data transfer speeds with different parallelisms. These are used by heuristic algorithms for task scheduling and resource allocation.

3.3 Heuristic Task Scheduling

Essentially, it is NP-Complete to make an allocation of resources for a flow of tasks, where the computation is tremendously intensive, especially when a large number of tasks and candidate resources are involved. Heuristic algorithms are preferable, among which the genetic algorithm (GA) is adopted. Derived from the life world with the mechanism of inheritance, aberrance, competition and selection, GA is believed to be promising to find satisfactory solutions, not necessarily the globally optimal ones, in a relatively short time.

It is the most important to set the evaluation index (i.e., fitness), which will be used to determine which individuals, called chromosomes, are to be reserved or distorted, and the individual with the best evaluation index will be kept as the optimal solution to our problem. Here the fitness of each chromosome is its throughput, and at that time we suppose that adequate data can be supplied, so as to separate task mapping and data supply. When calculating the throughput of each chromosome, the predicted task execution time on corresponding resources should be adopted, and much attention should be paid to task dependencies.

The coding of chromosome is another item of great importance, which reveals the possible form of the optimal solution to our problem. Our chromosomes in this allocation can be expressed as a vector:

$$chro = \{chro_i\}_{1\times n},$$

where each element of the vector stands for the allocated resources for task i from 1 to n. With online

measurement and prediction mentioned in subsection 3.2, taking into account execution time for each task, every chromosome in the candidate group is checked to find the optimal one. With the mechanism of inheritance, aberrance, competition and selection, after some generation of evolution, we can find at least a satisfactory chromosome, i.e., the solution for our problem.

Note that backlogs are not taken into fitness index, for excessive backlogs can be avoided with on-demand data streaming, including on-the-fly adjustment of transfer parallelism and repertory policy, without decreasing the total throughput.

3.4 On-demand Data Streaming

Data for each task have its own magnitude and the total available storage is limit. The data streaming will be stopped if the sum of magnitudes exceeds the total available storage and resumes when the backlog of data is processed. If there are too much data for a certain task, the data streaming will be intermittent rather than continuous. The data for a task is cleaned up after output data have been transferred to its subsequent tasks to save space for subsequent data. Tasks run constantly to perform data processing if there are waiting data, otherwise they are just idle.

Too much data occupy redundant resources, such as storage, network bandwidth, which is out of initial intention of data streaming applications; on the other hand, insufficient data supply will make tasks in a workflow lack of data to process, become idle and waste CPU cycles. On-demand data streaming is proposed in our work, which tries to supply data as much as required. Ideally, if data are streamed in a same speed as that are processed, data storage could be kept minimum. In our case, this is achieved by adjusting GridFTP parallelisms and controlling start and end times of data transfers using repertory strategies.

3.4.1 GridFTP parallelism tuning.

After resources are allocated to tasks, required data are streamed to corresponding tasks, which is another factor influencing the ultimate data throughput. GridFTP is applied as the data transfer protocol for cross-domain data replication. The GridFTP parallelism can be tuned to get optimal transferring performance. Here our goal is to guarantee data supply and get maximum throughput, meanwhile keep a minimum amount of medium data.

It is a non-trivial task to determine the proper amount of bandwidth to be allocated for data streaming applications. 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.

Monitoring the data processing tasks and the storage, we can get information on data requirements, denoted as S_{opt}. By transferring trace packages of data, data transfer speeds can be estimated with parallelisms of 1, 2, 4, 6 and 8, denoted as S_1, S_2, S_4, S_6 and S_8, respectively, where we have $S_1 < S_2 < S_4 < S_6 < S_8$ in general. Then it is transformed to a matchmaking problem to find the appropriate parallelism p, as the GridFTP parallelism parameter.

If $S_{opt} > S_8$, let p=8, and in our scenario this is just best-effort; if $S_{opt} < S_1$, let p=1; otherwise, we try to find the p which satisfies $S_{p-1} < S_{opt} \leq S_p$, $p \in \{2,4,6,8\}$. In the latter two cases, some repertory strategies should be applied to prevent data overflow in a long period of data processing.

A well-set parallelism may not always match data requirements over a long period, which makes it necessary to evaluate these parameters periodically. During a time interval, if GridFTP can not be used to get enough data, its parallelism should be set to its upper neighbor value, e.g., from 6 to 8. But it must be cautious to set the parallelism to a lower level unless redundant transferring speeds are 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.

Among the parameters of GridFTP 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 3. 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 3, the parallelism can be set to 4 or 6 to guarantee data supply with minimum bandwidth.

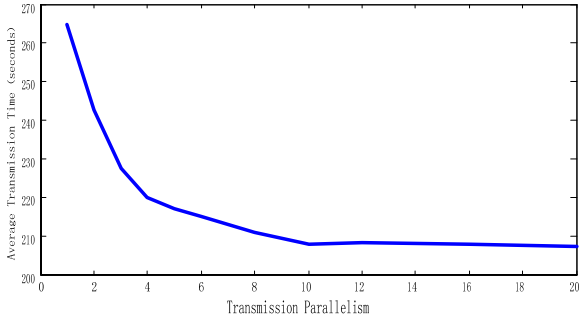

Figure 3. Comparison of data transfer times using different GridFTP parallelisms

3.4.2 Repertory strategies. 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. 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. 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.

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 varying between lower and upper limits.

For the sake of simplicity, the total amount of data items, including all the input items from remote sources and medium items of tasks are counted together to carry out repertory strategy. Status of streaming can be described as active and inactive, where the former means data are being transferred while the latter stands that no more data are supplied. A series of variables, named as *TS* (short for transfer status), can be defined to depict such status as following

$$TS^{(0)} = 1$$

$$TS^{(k)} = \begin{cases} 1, if\ TS^{(k-1)} = 1 and\ S^{(k)} < U \\ 0, \quad if\ S^{(k)} \geq U \\ 0, if\ TS^{(k-1)} = 0\ and\ S^{(k)} > L \\ 1, \quad if\ S^{(k)} \leq L \end{cases}, k=1,2,\ldots$$

where values 1 and 0 of $TS^{(k)}$ (k=0,1,2,…) stand for active and inactive status of data transfer respectively; U and L are the settled upper and lower limits of storage usage; $S^{(k)}$ is the total occupied storage of input items and medium items at interval; the upper label k means the k-th interval. Thus, given the upper and lower limits, measuring the occupied storage every interval will decide the transfer status, which is rather simple for implementation.

4 Performance Evaluation

A campus computational grid is being established in Tsinghua University (Beijing, China) which holds a large amount of supercomputers, personal computers and other special instruments. Globus toolkit 4.0.1 is being deployed to provide common grid services and a simple Certificate Authority has been established to sign certificates for hosts and users which will be used to establish a secure and transparent environment for data streaming applications. Among the campus computing grid, network file system has been established to be a public data holder, which can be accessed as if data are stored on local disks.

To verify our approach described in Section 3, we conduct an experiment for a data streaming application which contains a flow of 10 tasks, as shown in Figure 4, where there are dependencies between tasks, denoted by directed arrows. The nodes with character I and O stand for the input and output of the workflow, where the former means data streaming from remote source and the latter is the stage to collect the processing results.

Perhaps some explanation of this case is need.

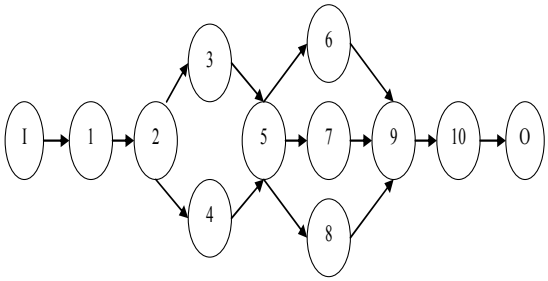

Figure 4. A Case Study

4.1 Workflow Deployment

With online measurement of a reasonable period of time, we get *average* execution times for each task on different resources in a matrix T:

$$T=\begin{bmatrix} 5 & 1 & 9 & 7 & 6 & 9 & 3 & 9 & 6 & 9 \\ 2 & 10 & 1 & 5 & 8 & 3 & 1 & 9 & 7 & 5 \\ 1 & 1 & 10 & 8 & 1 & 10 & 1 & 7 & 1 & 1 \\ 4 & 10 & 5 & 7 & 6 & 6 & 6 & 10 & 5 & 7 \\ 1 & 10 & 1 & 6 & 9 & 6 & 7 & 7 & 5 & 1 \\ 9 & 10 & 10 & 7 & 2 & 10 & 10 & 7 & 7 & 3 \\ 7 & 1 & 1 & 2 & 2 & 2 & 5 & 7 & 1 & 6 \\ 4 & 3 & 10 & 6 & 4 & 8 & 2 & 4 & 7 & 1 \\ 5 & 4 & 10 & 10 & 1 & 6 & 3 & 1 & 1 & 5 \\ 10 & 9 & 3 & 8 & 8 & 8 & 5 & 3 & 8 & 5 \end{bmatrix},$$

where t_{ij} represents the execution time in average of task i (i=1,2…10) on resource j (j=1,2…10) in a normalized unit.

Notice that the execution time for a task on a certain resource is not constant due to dynamic nature of grids due to changing workloads and competition with other applications, so the execution time matrix is different over time. The matrix T given above is just an average of such execution matrices for some sample routines, and in a long run what really accounts is just the average, not some peak values of execution time.

Now we concentrate on task scheduling and resource allocation to get the maximum throughput in certain time span. Initially the data streaming is not considered and will be discussed in the following subsection.

We set the size of population to be 40, mutation probability to be 0.1, total generations to be 100 and the crossover probability to be 0.2, as the parameters of our genetic algorithm. To evaluate the gross throughput, we set the period of time to be 1000 units. The chromosomes are coded as a vector:

$$chro_i=[*\ \ *\ \ *\ \ *\ \ *\ \ *\ \ *\ \ *\ \ *\ \ *],$$
$$i=1,2,\cdots,40$$

where * represents corresponding resources for tasks 1 to 10, respectively. Corresponding time for each task can be retrieved from the matrix T, and data throughput for one chromosome can be calculated as its fitness, which decides its fate, reserved or discarded.

After evolving for some generations, we get the optimal chromosome:

$$chro_{optimal}=[2\ \ 7\ \ 1\ \ 6\ \ 9\ \ 10\ \ 4\ \ 5\ \ 8\ \ 3],$$

which represents tasks 1 to 10 are allocated to resources 2, 7, 1, 6, 9, 10, 4, 8, 5, and 3, respectively, and the processing time vector is:

$$t=[1\ \ 1\ \ 1\ \ 6\ \ 5\ \ 3\ \ 2\ \ 4\ \ 1\ \ 3],$$

where $t(i)=T(i,chro_{optimal}(i)),\ i=1,2,\cdots,10$.

4.2 On-demand Data Streaming

For grid data streaming applications, data streaming and processing interact with each other, and the former increases data files in the NFS, while the latter decreases the number since data files will be cleaned up after being processed. At any time the current number of data files reflects combined effectiveness of data streaming and processing. If we have relatively large data storage or the processing speed is high enough, repertory strategies are not required; but, if storage is small or the computational resources are not powerful enough, some repertory strategies will be indispensable to avoid data overflow. We will discuss these two situations below.

4.2.1 Large storage with performance bottleneck without repertory strategies. In this experiment, the GridFTP parallelism is set to be the highest 8 at first and then decreased to 2. The backlog of data for each task and overall data throughput of the workflow in the number of data files are measured as shown in Table 2.

Table 2. Experimental Result I

p	Backlog of Data									Data Throughput
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
8	0	334	335	0	332	0	0	0	0	165
2	0	84	85	0	85	0	0	0	0	164

The final data throughput is 165 when the GridFTP parallelism is set to 8 with the highest data streaming speed. It can be observed that backlogs of data at tasks s_2, s_3 and s_5 are also very high. This can be explained by the difference of data processing speeds between dependent tasks. For example, task s_2 has two subsequent tasks, s_3 and s_4. s_2 has a much higher data processing speed than s_4 according to the processing time vector t scheduled in Section 4.1. Although s_3 can process data as fast as s_2, output data of s_2 can not be cleaned up until also received by s_4, which causes a large backlog of data at s_2. There are similar situations at tasks s_3 and s_5. So for an application with a data streaming workflow, task scheduling and resource allocation has to consider balance between dependent tasks, otherwise data processing bottlenecks would result in low application performance and resource utilization.

The GridFTP parallelism is decreased from 8 to 2, which dramatically reduces originate data streaming speed. Experimental results included in Table 2 shows that the same final data throughput is achieved with much less backlogs of data at intermediate tasks. So it is not the case that the faster data are streamed the better, since too fast data transfer would result in backlogs of data, which could not be processed in time and has to consume unnecessary storage and bandwidth, instead of increasing the final data throughput.

4.2.2 Large storage without performance bottleneck without repertory strategies. The experimental results for this scenario are included in Table 3.

Table 3. Experimental Result II

p	Backlog of Data									Data Throughput
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
8	0	167	168	0	166	83	84	0	0	248
2	0	0	0	0	0	123	123	0	0	124

It is obvious that tasks s_4 and s_5 are bottlenecks of the whole data processing workflow in Figure 4. In this experiment, we increase processing speeds of these tasks by allocating more powerful processors to them.

Table 3 shows that eliminating bottlenecks increase the ultimate data throughput dramatically (from 165 to 248), compared with results included in Table 2. And the lower data streaming speed starve the whole flow of tasks, leading to a much lower final data throughput (124). With a relatively fast data processing workflow, a too slow data streaming support would also become a bottleneck in terms of the ultimate data throughput and lead to idle task processes and lower resource utilization.

While s_4 and s_5 are not bottlenecks any more, a new bottleneck s_8 appears as indicated in Table 3. The result is conformed to the processing time vector t scheduled in Section 4.1. The low performance of s_8 leads to backlogs of data at s_6 and s_7. This is because s_6, s_7 and s_8 are parallel branches and parents of s_9. s_9 has to consume output data of s_6, s_7 and s_8 simultaneously. If s_8 is performed slower than s_6 and s_7, output data of s_6 and s_7 are accumulated and become backlogs. So task scheduling and resource allocation has to consider balance among parallel tasks.

Backlogs of data at s_2 over time is also illustrated in Figure 5. It is obvious that storage aware resource allocation can eliminate bottlenecks, dramatically reduce the backlog of data, and improve the final data throughput.

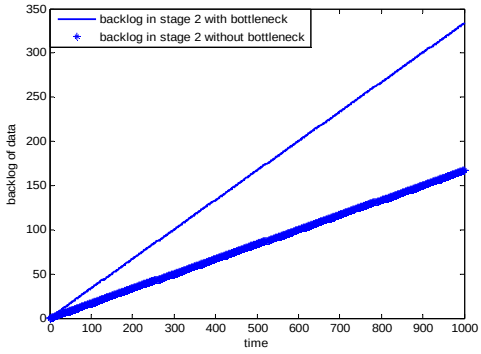

Figure 5. Comparison of Backlogs of Data at Task s_2 (p=8)

It is also shown in Figure 5 that the backlog of data at task s_2 increases approximately linearly over time, which does not scale well for long running data streaming applications. This can be improved if repertory strategies are applied in the next experiment.

4.2.3 Limit storage with repertory strategies applied. Based on the experiment carried out in Section 4.2.1, repertory strategies described in Section 3.4.2 is applied to avoid data overflow in this experiment. If the number of data files in storage reaches the upper limit, data transfers are stopped until the data storage is decreased to the lower limit. Corresponding results are included in Table 4.

Table 4. Experimental Result III

p	Backlog of Data									Data Throughput
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
8	0	12	13	0	11	0	0	0	0	165
2	0	11	12	0	10	0	0	0	0	164

Compared with results in Table 2, the same data throughput is achieved with much less backlogs of data. The data processing workflow is executed without unnecessary backlogs of data. In this case, data streaming is configured to provide data in an on-demand manner. As a result, backlogs of data at each task do not increase linearly over time as illustrated in Figure 6. Note that only 100 time units of data are included in Figure 6, instead of 1000 in Figure 5.

In Figure 6, data are no longer accumulated over time since originate data streaming stops at the upper limit once a data overflow is detected. This mechanism only works when data transfer speed is higher than average data processing speed of the workflow. In the $p=2$ situation of the experiment in Section 4.2.2, when the workflow already starves for data, the repertory strategy cannot be applied

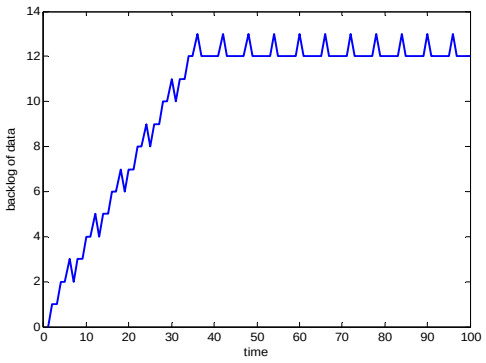

Figure 6. Backlogs of Data with Repertory Strategies Applied at Task s_2 (p=8)

4.2.4 Dynamic adjustment of parallelism. Usually, in grid, there are multiple applications (i.e., workflows) compete for limited resources, mainly the bandwidth. The overall principle is to guarantee *just enough* rather than redundant data supply for each workflow so as to alleviate congestion of network. Parallelism of data transfers is adjusted on the fly based on prediction, so as to match data supply and processing as much as possible. Results provided in subsection 4.2.1, 4.2.2 and 4.2.3 just show the influence of fixed parallelism in the evaluation period of time, while in this subsection, we will show how the on-the-fly parallelism of each data transfer should be adjusted.

Four workflows are considered simultaneously. Their processing speeds gotten by prediction based on measurement are shown in Figure 7, and the corresponding parallelism (denoted as p) of each data transfer is shown in Figure 8. For workflow 1, its processing speed is rather high, so data are being supplied with the max parallelism constantly, just in the best-effort way; for the rest workflows, data transfers are intermittent rather than continuous due to repertory strategy with appropriate parallelism. The blank areas in Figure 8 mean that data supply is not active, i.e., no more data are supplied at the corresponding time.

5. Technical Discussions

Several technical aspects of research are coupled with each other in this work:

Grid computing. The environment is developed in context of grid computing, focusing on cross-domain resource sharing within virtual organizations. In this work, data transfers are performed using GridFTP with on-the-fly tuning capability of parallelism.

Data streaming. A specific type of applications is considered in this work. These applications require large volumes of data fed continuously to flows of sequential / parallel tasks. Intermediate data are also required to be streamed among dependent tasks. Data streaming and processing are carried out concurrently to improve overall system performance in this work.

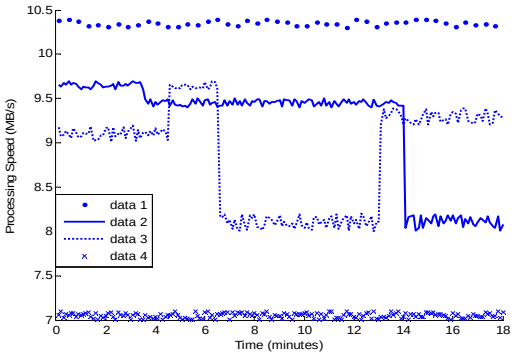

Figure 7. Predicted processing speeds

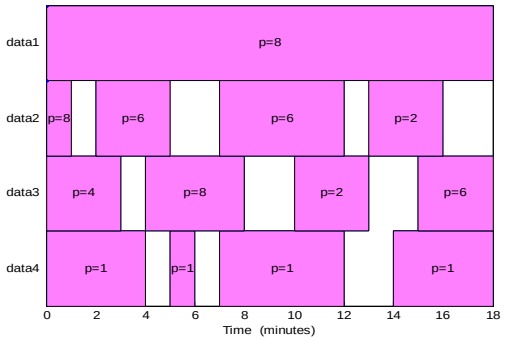

Figure 8. Dynamically adjusted parallelism

Measurement and prediction. Performance predication provides a basis for task scheduling and resource allocation. In this work, a *GM(1,1)* method is adopted to get performance information on task execution time on grid resources.

Task scheduling and resource allocation. Data streaming applications bring new challenges to task scheduling and resource allocation. In this work, the genetic algorithm is used to solve the NP-Complete problem of mapping tasks to available resources using heuristic search for optimal solutions.

Storage management. Storage awareness in grid enabling data streaming applications is the major focus of this work since such applications always involve large volumes of data in time series. On-demand data provision and just-in-time cleanup is proposed in this work to address the challenge.

Several experiments are carried out in this work and our discoveries are summarized below.

- *The the-faster-the-better principle does not apply to data streaming applications in some situations*. For example, if data streaming

speed is higher than processing speed at an average, a backlog of intermediate data occurs at bottlenecks of data processing workflows, which leads to unnecessary storage usage. Applying GridFTP with on-the-fly adjustable parallelism, we can approach the optimal speed as much as possible.

- *Balances among dependent tasks of a workflow become essential to achieve high performance*. Sequential / parallel tasks should have similar data processing speed, otherwise system bottlenecks and backlogs of data occur which consumes unnecessary storage. In a flow of tasks, while execution makespan is focused in traditional task scheduling and resource allocation research, our work focuses more on ultimate data throughput since data streaming applications always run continuously.
- *Implementation of data streaming applications not necessarily requires large data storage capability*. Storage aware resource allocation using repertory strategies is proposed in this work so that applications can gain high data throughput with relatively small storage.

6. Related Work

There are existing grid resource allocation infrastructures, e.g. Legion [10], Globus and Condor, with limit task and workflow scheduling supports. Issues in task scheduling for grid workflows are investigated in [11] and [12] though no data storage constraint is considered. While there are existing storage management grid middleware, e.g. SRB [13] and SRM [14], no specific data streaming supports are provided in these tools.

This work is focused on a resource management and scheduling infrastructure for grid data streaming applications. Some existing work on data streaming management are derived from database management systems, e.g. STREAM [15], Aurora [16], NiagaraCQ [17], StatStream [18], and Gigascope [19]. Most of data streaming research in context of grid computing [20][21][22][23] are application specific and scheduling issues are not addressed. The following two systems are most related to the work described in this paper:

- *Streamline* [24], a scheduling heuristic based on Globus, is designed to adapt to the dynamic nature of a grid environment and varying demands of a streaming application. Streamline is not storage aware since no storage scheduling issues are considered. Our implementation is *storage aware* since CPU-rich grid resources maybe storage-limit.
- *Pegasus* [25] has the most similar motivation with the work described in this paper. While Pegasus handles data transfers, job processing and data cleanups in a pipeline manner, in our environment, data streaming, processing and cleanup are *concurrent*. By carefully optimizing data streaming, our environment makes required data available on-demand and just-in-time, which dramatically reduces storage requirements for executing grid data streaming workflows.

7. Conclusions and Future Work

Data streaming applications are becoming very popular especially with the development of open grid environments and resources. In order to utilize remote grid resources that are CPU-rich and storage-limit, data streaming has to be performed concurrently with data processing with carefully scheduled storage usage.

Experimental results described in this work show that in order to achieve high data throughput with limit storage usage for data streaming applications, several principles have to be considered that are different from traditional job scheduling and resource allocation. For example, system performance is improved if data streaming and processing are concurrent. Also data streaming does not necessarily lead to large amount of storage usage, since if data are streamed on-demand instead of as-fast-as-possible, and there is a good balance of data processing speeds among dependent tasks of a workflow, that is to say, if data streaming and processing are storage aware, high data throughput can be still achieved with relatively small storage.

Ongoing work include grid enabling several real applications using our data streaming environment, further performance evaluation of algorithms proposed in this work in real time execution of workflows, and extension of current approach to work in an evolving manner so that grid dynamism can be addressed and adaptability of our environment can be improved.

Experiments discussed in this work only focus on individual data streaming applications. This is not the case in real situation where there could be multiple applications sharing grid resources in the same virtual organization. More constraints have to be considered if multiple flows of tasks are considered, e.g. network bandwidth. Task scheduling and resource allocation will also become much more complicated if multiple data processing workflows are sharing data streams

and competing resources. These issues will be addressed in our future work.

Acknowledgement

This work is supported by National Science Foundation of China (grant No. 60803017), Ministry of Science and Technology of China under the national 863 high-tech R&D program (grants No. 2006AA10Z237, No. 2007AA01Z179 and No. 2008AA01Z118), Ministry of Education of China under the program for New Century Excellent Talents in University and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, and the FIT foundation of Tsinghua University.

References

[1]. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, 1998.

[2]. E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda, "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists", *Proc. 11th IEEE Int. Symp. on High Performance Distributed Computing*, pp. 225-234, 2002.

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

[4]. J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[5]. 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.

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

[7]. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith and S. Tuecke, "A Resource Management Architecture For Metacomputing Systems", *IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 62-82, 1998.

[8]. W. Zhang, J. Cao, Y. Zhong, L. Liu and C. Wu, "An Integrated Resource Management and Scheduling System for Grid Data Streaming Applications", *Proc. 9th IEEE/ACM Int. Conf. on Grid (Grid 2008)*, pp.258-265, Tsukuba, Japan.

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

[10]. S. J. Chapin, D. Katramatos, J. Karpovich and A. S. Grimshaw, "The Legion Resource Management System", *Job Scheduling Strategies for Parallel Processing*, Springer Verlag, pp.162-178, 1999.

[11]. R. Sakellariou and H. Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems", *Proc. 13th IEEE Heterogeneous Computing Workshop*, 2004.

[12]. D. P. Spooner, J. Cao, S. A. Jarvis, L. He, and G. R. Nudd, "Performance-aware Workflow Management for Grid Computing", *The Computer J.*, Special Focus - Grid Performability, Vol. 48, No. 3, pp. 347-357, 2005.

[13]. A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S. Chen, and R. Olschanowsky, "Storage Resource Broker - Managing Distributed Data in a Grid", *Computer Society of India Journal*, Special Issue on SAN, Vol. 33, No. 4, pp. 42-54, 2003.

[14]. A. Shoshani, A. Sim, and J. Gu, "Storage Resource Managers: Middleware Components for Grid Storage", *Proc. 19th IEEE Symp. on Mass Storage Systems*, 2002.

[15]. 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.

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

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

[18]. 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.

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

[20]. R. Kuntschke, T. Scholl, S. Huber, A. Kemper, A.

Reiser, et. al, "Grid-Based Data Stream Processing in e-Science", *Proc. 2nd IEEE Int. Conf. on e-Science and Grid Computing*, 2006.

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

[22]. 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.

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

[24]. 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.

[25]. 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", *Proc. 7th IEEE Int. Symp. on Cluster Computing and the Grid*, Rio de Janeiro, Brazil, pp. 401-409, 2007.