Grid Resource Management and Scheduling for Data Streaming Applications

Wen Zhang[a], Junwei Cao[b,c*], Yisheng Zhong[a,c], Lianchen Liu[a,c], and Cheng Wu[a,c]
[a]*Department of Automation, Tsinghua University, Beijing 100084, China*
[b]*Research Institute of Information Technology, Tsinghua University, Beijing 100084, China*
[c]*Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China*

Abstract

Data streaming applications bring new challenges to resource management and scheduling for grid computing. Since real-time data streaming is required as data processing is going on, integrated grid resource management becomes essential among processing, storage and networking resources. Traditional scheduling approaches may not be sufficient for such applications, since usually only one aspect of grid resource scheduling is focused. In this work, an integrated resource scheduling approach is proposed and coordinated resource allocation of CPU cycles, storage capability and network bandwidth is implemented. Resource allocation is performed periodically with updated information on resources and applications and heuristic search for optimal solutions is used to assign various resources for running applications simultaneously. Performance metrics considered in this work include data throughput and utilization of processors, storage, and bandwidth, which are actually tightly coupled with each other when applied for grid data streaming applications. Experimental results show dramatic improvement of performance and scalability using our implementation.

Keywords: Grid Computing; Data Streaming; Resource Management; Genetic Algorithm

1. Introduction

Data streaming applications are becoming more popular recently, such as astronomical observations, large-scale simulation and sensor networks, which brings new challenges to grid resource management. Characteristics of such applications include that (1) they are continuous and long running in nature; (2) they require efficient data transfers from distributed data sources; (3) it is often not feasible to store all the data in entirety for later processing because of limited storage and high volumes of data to be processed; (4) they need to make efficient use of high performance computing (HPC) resources to carry out computation-intensive tasks in a timely manner. Grid computing [1] paves a new way to enable such applications with cross-domain resource sharing and service integration supports.

When there is a shortage of CPU processing capability located at data sources, there is a requirement that data have to be streamed to computational resources for processing. For example, LIGO (Laser Interferometer Gravitational-wave Observatory) [2][3] is generating 1TB scientific data per day and trying to benefit from processing capabilities provided by the Open Science Grid (OSG) [4]. Since most OSG sites are CPU-rich but storage-limited with no LIGO data available, data streaming supports are required in order to utilize OSG CPU resources. In such a data streaming scenario, data should be pulled rather than pushed to the computational system in the form of streams of tuples, and processing is continuously executed over these streams as if data were always available from local storage. What's more, data arrival rates must be controlled to match the processing speeds to avoid waste of computational capacity or data overflow. Meanwhile, processed data have to be cleaned up to save space for the subsequently coming data.

Grid enabled data streaming applications require management of various grid resources, e.g. CPU cycles, storage capability and network bandwidth. In this paper, an integrated resource management and scheduling system for grid data streaming applications is developed to improve data throughput, processor utilization, storage usage and bandwidth utilization in a coordinated way. When a new application arrives, admission control is invoked to decide whether to start or queue it. Accepted applications are allocated with appropriate resources at the end of each scheduling period, together with already running ones. A heuristic approach, genetic algorithm [5] (GA), is applied

* Corresponding author. E-mail address: jcao@tsinghua.edu.cn

to find satisfactory resource allocation scheme in the given scheduling period with updated information of resources and applications. The scheduling is evolving periodically with updated status of resources and applications since the grid is a shared environment where resources are not dedicated. Based on the Globus toolkit [6], the system is able to discover and manage resources geographically distributed and belonging to different management domains in a transparent and secure way. Evaluation results show excellent performance and scalability of this system.

The rest of this paper is organized as follows: Section 2 provides a formal representation of the optimization issue with predefined performance metrics; corresponding algorithms are elaborated in Section 3; performance evaluation results are illustrated in Section 4; related work are discussed in Section 5; and Section 6 concludes the paper.

2. Grid Data Streaming – Problem Statement

As mentioned above, an integrated resource management and corresponding scheduling algorithms are required to make full resource utilization while keeping optimal performance of each data streaming application. The approach tries to accommodate as many applications as possible simultaneously to make the best use of resources in preconditions that requirements of each application can also be met. In this section, the schedule issue is described in a formal way and performance metrics are defined.

2.1. Formal Representation

A resource pool R described in this work include processors (P), storage (S) and network bandwidth (B) that have to be allocated to data streaming applications in an integrated manner. Suppose n is the total number of processors P in the resource pool and there are m applications (A) for data streaming and processing.

$$R = \{P, S, B\}$$

$$P = \{p_i \mid i = 1,2,.....,n\}$$

$$A = \{a_j \mid j = 1,2,.....,m\}$$

Let s and b be the total storage space S and network bandwidth B, respectively. In general, the sum of processors, storage and bandwidth allocated to each application cannot exceed the total available resources.

$$\sum_{j=1}^{m} n_j \le n$$

$$\sum_{j=1}^{m} s_j \le s$$

$$\sum_{j=1}^{m} b_j \le b$$

For each application a_j, there is a corresponding minimum requirement of resources that has to be met for the application to be executed.

$$R_j^{req} = \{n_j^{req}, s_j^{req}, b_j^{req}\}$$

$$n_j^{req} \le n_j \le n, j = 1,2,.....,m$$

$$s_j^{req} \le s_j \le s, j = 1,2,.....,m$$

$$b_j^{req} \le b_j \le \min(b, b_j^{\max}), j = 1,2,.....,m$$

All of above constraints have to be met during resource scheduling and allocation for data streaming applications. Note that the bandwidth allocated to the application a_j is constrained by both available bandwidths locally b and remotely at the data source end b^{max}_j. The major challenge is that the three different types of resources are correlated to each other inherently in deciding the performance of a scheduling and allocation scheme. In the next section, major performance metrics considered in this work are described

2.2. Performance Metrics

There are many aspects of performance criteria when evaluating resource allocation schemes. Specifically, for data streaming applications, data throughput, the amount of data streamed and processed during a given period of time, is the most important. Other performance metrics that have to be considered simultaneously are resource utilization and scheduling scalability.

Suppose an evaluation period includes l time units. A time unit t is a predefined minimum time period, based on which all resource scheduling and allocation are carried out. Let b^{usg}_{jk} (j=1,2,……,m;k=1,2,……,l) be the bandwidth usage of the application a_j during the k^{th} time unit and s^{usg}_{jk} (j=1,2,……,m;k=1,2,……l) be the storage usage at the beginning of the k^{th} time unit. Note that actual resource usage of an application is usually different from corresponding resources allocated to an application. We can calculate the total data throughput TP as follows:

$$TP_{jk} = b^{usg}_{jk} t + s^{usg}_{jk} - s^{usg}_{j(k+1)} (j = 1,2,....., m; k = 1,2,....., l)$$

$$TP_j = \sum_{k=1}^{l} TP_{jk} = \sum_{k=1}^{l} b^{usg}_{jk} t + s^{usg}_{j1} - s^{usg}_{j(l+1)} (j = 1,2,....., m)$$

$$TP = \sum_{j=1}^{m} TP_j = \sum_{j=1}^{m} \sum_{k=1}^{l} b^{usg}_{jk} t + \sum_{j=1}^{m} (s^{usg}_{j1} - s^{usg}_{j(l+1)})$$

The total data processing throughput is the difference of storage usage plus all data streamed into the system during the given period. This is based on the assumption that just-in-time data cleanup is enabled and all processed data are cleaned up from storage at the end of each time unit. If the evaluation period covers all the makespan of an application, it is obvious that and s^{usg}_{j1} are $s^{usg}_{j(l+1)}$ are both zero and the total data processing throughput for a given application can be represented purely via bandwidth usage. To simplify the problem, we assume that data throughput of each application TP_j are *comparable* with each other, so that a sum up of all TP_j can be used to represent the overall data throughput. If this is not the case in a real environment, some normalization has to be performed to weight data throughputs of different applications in terms of data throughput.

Resource utilization is another concern when enabling data streaming applications. It is straightforward to calculate storage and bandwidth usage percents of the k^{th} time unit as follows:

$$US_{jk} = \frac{s^{usg}_{jk}}{s} (j = 1,2,....., m; k = 1,2,....., l)$$

$$US_k = \frac{\sum_{j=1}^{m} s^{usg}_{jk}}{s} \le 1 (k = 1,2,....., l)$$

$$US = \frac{\sum_{k=1}^{l} \sum_{j=1}^{m} s^{usg}_{jk}}{ls} \le 1$$

$$UB_{jk} = \frac{b^{usg}_{jk}}{b} (j = 1,2,....., m; k = 1,2,....., l)$$

$$UB_k = \frac{\sum_{j=1}^{m} b^{usg}_{jk}}{b} \le 1 (k = 1,2,....., l)$$

$$UB = \frac{\sum_{k=1}^{l} \sum_{j=1}^{m} b^{usg}_{jk}}{lb} \le 1$$

The utilization of CPU cycles can be calculated indirectly via storage usage, since for data streaming applications, it can be assumed the allocated processor is busy when there are available data in local storage, and

idle when no data available locally. Suppose that P_{jk} is the set of processors that are allocated to the application a_j during the k^{th} time unit. Let M_{ik} be a 2D array to identify if the processor p_i is busy or idle at the k^{th} time unit.

$$M_{ik} = \begin{cases} 1 & if \quad \forall j \quad p_i \in P_{jk} \quad and \quad s_{jk}^{usg} > 0 \\ 0 & if \quad \exists j \quad p_i \notin P_{jk} \quad or \quad s_{jk}^{usg} = 0 \end{cases} (i = 1,2,.....,n; k = 1,2,.....,l)$$

The processor usage percent is calculated as follows:

$$UP_i = \frac{\sum_{k=1}^{l} M_{ik}}{l} \le 1 (i = 1,2,.....,n)$$

$$UP = \frac{\sum_{i=1}^{n}\sum_{k=1}^{l} M_{ik}}{nl} \le 1$$

The resource management and scheduling issue for grid data streaming applications can be transformed into an optimization problem:
P.

$$\max TP$$
$$\max UP$$
$$\min UB$$

s.t.

$$UP_i \le 1 (i = 1,2,.....,n)$$
$$US_k \le 1 (k = 1,2,.....,l)$$
$$UB_k \le 1 (k = 1,2,.....,l)$$

where first two goals are to process more data and match data processing and streaming capability as much as possible while the third one is to utilize bandwidth in an economic way to avoid congestion. These three goals conflict in nature, and some tradeoffs have to be made. Currently, we focus more on the overall data throughput. Algorithms provided in the next section is processing-, storage-, and congestion-aware, so the last two goals can be maintained in most cases. Note that storage usage is not included in optimization goals because storage usage does not affect the ultimate throughput, but adequate storage will indeed increase the robustness of data processing. Available processors, storage and bandwidth are considered as constraints.

3. Resource Scheduling and Allocation – Key Algorithms

There are two steps for resource scheduling: admission control is performed to decide whether a new application is started, according to its resource requirement and current status of available resources in the computing pool; the GA is performed periodically to improve resource utilization and meet requirements of active applications in an evolving way. Resource allocation is performed iteratively together with periodical scheduling of key parameters.

In this section, key components of this resource management and scheduling scheme are described in details. The overall flow chart for such a scheduling process is illustrated in Figure 1.

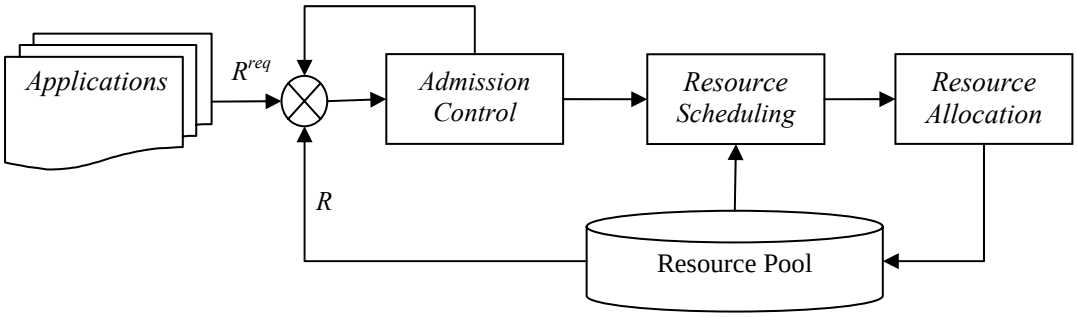


Figure 1. The Flow Chart of Resource Scheduling and Allocation for Data Streaming Applications

A coming application has to specify explicitly its requirements of resources, R^{req}. Available resources R in the resource pool are monitored in real-time. Both R^{req} and R are input to the admission control module to decide whether or not the coming application should be accepted and put to the active application set. Resource scheduling only works on active application set periodically to produce scheduling parameters. In this work, the GA is adopted as an evolving method to absorb dynamically changing resource and application status. Resource allocation takes scheduling parameters as inputs and generates final allocation schemes. Resource allocation occurs iteratively with a much higher frequency than resource scheduling to improve the overall system performance. These are described in details below.

3.1. Admission Control

It is obvious that the resource pool in general cannot support infinite applications simultaneously, and too many applications will lead to fierce resource competition, which may decrease overall processing efficiency as a whole. The mechanism of admission control plays an essential role in our system for resource management.

When a new task is submitted, the admission control decides to accept it instantly or just keep it in a waiting queue and resubmit it in future. This decision is made according to the usage status of resources and application requirements. Each task can specify its minimum requirement of resources, e.g., it needs some processors of certain types, minimum bandwidth and storage. An XML schema is developed for application requirement specification.

Applications can specify CPU type, bandwidth and storage requirement. The system checks up current available resources. For example, an application compiled on X86_64 cannot run on I386 processors, so not every processor is suitable for the application. Suppose that the number of those available X86_64 processors is larger than n^{req}_j, and unallocated storage and bandwidth are both larger than s^{req}_j and b^{req}_j, respectively, the task can be immediately put to be active for execution. If any of resources is not available, the task would just be kept in the waiting queue.

Applications in the queue are called new eligible applications (NEAs). NEAs can have different priorities for resource scheduling and allocation. NEAs are classified into different groups according to different priorities, and in each group, the first-come-first-serve (FCFS) strategy is applied.

3.2. Resource Scheduling

As CPU, bandwidth and storage are integrated as a whole in data streaming applications, there must be a so-called optimal operation point (OOP) to make balance among resource usage. The OOP defines the expected bandwidth, computing power, and granularity of data streaming (storage usage), which simultaneously maximizes uses of bandwidth and CPU power. Our scheduler benefits greatly from using an optimal combination of resources, instead of making independent scheduling schemes for each type of resources.

Essentially, integrated scheduling of resources for applications is a NP-Complete problem, and in this work the GA is adopted to find satisfactory, not necessarily optimal, solutions in a relatively short time. The GA is required to recalculate scheduling parameters in each scheduling period, with updated status information of resources and applications.

Using the GA, a single chromosome is composed with three parts of scheduling parameters, leading to an allocation scheme of processors, storage and bandwidth, respectively. The g^{th} generation of chromosome can be represented as follows:

$$CHROM_g = \{p_{jg}, s_{jg}, \alpha_g, \beta_g, \rho_g, \mu_{jg} \mid j = 1,2,....., m\}$$

Similar to definitions in Section 2.2, suppose that P_{jg} is the set of processors that are allocated to the application a_j during the g^{th} generation of GA evolving. To simplify the problem, each application just needs one processor, so p_{jg} is used above instead of P_{jg}. For an application, it is expected to run on a fixed processor till it is finished, or put it another way, no migration from one processor to another occurs during a task execution. Only parts of p_{jg} for new applications are involved in the GA evolving and processors assigned previously are fixed to be allocated to its existing application. s_{jg} represent the maximum storage allocated to the application a_j. For a certain application, its lower limit of storage has to be larger than s^{req}_j and can be set to be a proportion of s_{jg}. Details are included in the next section on resource allocation. Similarly, scheduling parameters above, α, β, ρ and μ_j (j=1,2,……,m) are corresponding to bandwidth allocation. How these parameters are associated with a bandwidth allocation scheme is also described in details in the next section. Three parts of a chromosome evolve independently with their own rules, decreasing computational complexity and avoiding meaningless heuristic searches.

The evaluation index, or fitness, of each chromosome is set to be data throughput, i.e., the amount of data processed in a scheduling interval. As mentioned before, we consider all the data for different applications equally. In calculating its throughput, information on applications and resources has to be updated and performance prediction is enabled using historical information. Given a chromosome with scheduling parameters, with historical performance information and some priori of resources and applications, data throughput in a scheduling interval can be calculated using formulations in Section 2.2. In a scheduling period, scheduling parameters are initiated from its direct foregoing period. During the evolution, two chromosomes are crossed to generate two new ones for the next generation, and genetic mutation happens in some chromosomes with a given probability. The chromosome that leads to the highest data throughput value can be achieved and corresponding scheduling parameters are used to generate a resource allocation scheme in a scheduling period.

Although it is hard to find the global OOP since applications and resources are varying constantly and it is not easy to define expected processors, bandwidth and storage precisely, evolving searching capability of the GA guarantees a satisfactory solution for each scheduling period. Allocation algorithms for processors, storage and bandwidth are given below.

3.3. Resource Allocation

Given a set of correlated scheduling parameters generated by the GA described above, resource allocation schemes can be achieved using the method given in this section. While scheduling parameters are fixed within each scheduling period, resource allocation schemes can still change, for instance, bandwidth allocation is an iterative process.

3.3.1. Processor Allocation

As mentioned above, processor assignment is a match making process. Both applications and resources can specify their own requirements. Processors can be classified into several groups according to their architectures. Similar processors in the same group may also have different frequencies that may lead to different data processing performance. NEAs can also be organized into several groups with different priorities. In each group, the selecting principle is FCFS.

Matchmaking is carried out to find appropriate processors for applications. The processors whose characteristics do not conflict with the application requirements form a candidate set. Then applications with higher priorities find their matched processors first. In a *CHROM*, p_j is the No. of the assigned processor for each application. In different generations of evolving in a given scheduling period, processor assignments of chromosomes in successive generations are independent to guarantee all possible assignments can be covered. As we have mentioned, no migration of applications exist, so in each scheduling period, the algorithm is only required to allocate processors for the NEAs.

3.3.2. Storage Allocation

The overall principle for storage allocation is to make full usage of storage to increase robustness while getting ready for new coming applications. If there are only a few applications running in the resource pool, the storage allocated for each application can be set to a high value. While the number of applications increases, allocated storage for each application may be decreased. So quotas for each application can be scalable accordingly, and there must be some margin of storage for potentially new-coming applications.

① Initialization: as supposed, there are m applications in the pool, to generate m random numbers, $r_j \in (0, 1)$, j=1,2,……,m. Calculate each quota, q_j as follows:

$$q_j = \frac{r_j}{\sum r_j} (j = 1,2,....., m)$$

② If $s_j=q_j s \geqslant s^{req}_j$, reserve these numbers for initially allocated storage for the application a_j; else, repeat step ① until $s_j \geqslant s^{req}_j$ (j=1,2,……,m) hold true, where s^{req}_j is the minimal required storage of application a_j as defined in Section 2.1.

③ Repeat ① and ②, until all the storage allocation schemes are set for the chromosomes in a population, and these would be initial values for the first generation.

④ Chromosome crossing: Two chromosomes cross to generate new ones for the next generation as follows:

$$\varepsilon_{jg} s_{jg} + (1-\varepsilon_{jg}) s_{j'g} = s_{j(g+1)}$$
$$(1-\varepsilon_{jg}) s_{jg} + \varepsilon_{jg} s_{j'g} = s_{j'(g+1)}$$

where $0<\varepsilon_{jg}<1$ is a random number evenly distributed in (0,1), and the other chromosome, j', is selected at random.

⑤ Repeat ④, until all the generations are covered.

A storage allocation scheme includes both the upper limit and lower limit values. As mentioned in Section 3.2, the lower limit of storage has to be larger than s^{req}_j and can be set to be a proportion of s_{jg}. Lower and upper limits are mainly used as thresholds to control start and end times of data streaming: 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.

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. 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.

3.3.3. Bandwidth Allocation

Bandwidth allocation plays an important role in the whole resource allocation scheme, for appropriate bandwidth is indispensable to guarantee data streaming for applications to make them run constantly. Different from traditional bandwidth allocation, our scheme is storage aware, i.e., data streaming may be intermittent rather than continuous to avoid data overflow, for allocated storage for each application is limited. When the storage is full of data, streaming is halted for a while until some data have been processed and cleaned up so that storage is released for more data. At any moment, the amount of data in storage for each application is determined by both data streaming and clean-up processes.

A utility function is introduced $U_j(b_j)$ when its data is streamed with allocated bandwidth b_j. $U_j(b_j)$ should be concave, continuous, bounded and increasing in the interval $[b^{req}_j, b]$. Note that it is not necessary that identical utility functions are chosen for all applications. We try to maximize the sum of the utilities of all the applications, maintaining fairness among them.

Due to the usage status of storage and repertory policy with lower and upper limits of storage allocation, there are two possible states for each application a_j at any time, i.e., active and inactive, which indicates if a data streaming is on or off. Let s_{jk} and λs_{jk} $(0<\lambda<1)$ be upper and lower limits of storage allocation. λ is a predefined fixed proportion of storage allocation between 0 and 1. Let A_{jk} be the state identity for the application a_j at the k^{th} step. A_{jk} can be initialed as active (1) and evolves as follows:

$$A_{j(k+1)} = \begin{cases} 1 & if \quad A_{jk} = 1 \quad and \quad s^{usg}_{jk} < s_{jk} \\ 0 & if \quad s^{usg}_{jk} \geq s_{jk} \\ 0 & if \quad A_{jk} = 0 \quad and \quad s^{usg}_{jk} > \lambda s_{jk} \\ 1 & if \quad s^{usg}_{jk} \leq \lambda s_{jk} \end{cases}$$

An iterative optimization algorithm is proposed in [7] and its convergence is analyzed, but it is required to be congestion-aware, which is hard to be satisfied in the wide-area Internet. According to our situation, we make some modification upon it as follows:

$$b_{j(k+1)} = \begin{cases} \left[b_{jk} + \alpha_k U'(b_{jk}) \right]_j & if \quad A_{jk} = 1 \quad and \quad \sum_{A_{jk}=1} b_{jk} \leq \rho_k b \\ \left[\beta_k b_{jk} \right]_j & if \quad A_{jk} = 1 \quad and \quad \sum_{A_{jk}=1} b_{jk} > \rho_k b \\ 0 & if \quad A_{jk} = 0 \end{cases}$$

Here, b_{jk} is the bandwidth allocation for the application a_j at the k^{th} step of iterations. Note that there are many small iteration steps during a scheduling period. α_k, β_k and ρ_k are all positive sequences. ρ_k is the so-called safety coefficient to avoid bandwidth excess, where $\rho_k \in (0, 1)$, i.e., there is some margin from the full use of total bandwidth for flexibility and robustness. For the sake of convenience, they are usually substituted as a fixed value within multiple iteration steps of a scheduling period and only evolve when GA is applied to generate new scheduling parameters when a new scheduling period starts, as described in Section 3.2. $[\cdot]_j$ denotes a projection on the application a_j and can be calculated as:

$$[x]_j = \min\left(b, b_j^{\max}, \max\left(b_j^{req}, x\right)\right)$$

U'(.) is the sub gradient of

$$U(\cdot) = \sum_{j=1}^{M} U_j\left(b_{jk}\right)$$

and

$$U'\left(b_{jk}\right) = \frac{\partial U(\cdot)}{\partial b_{jk}}$$

A popular utility function can be expressed as:

$$U_j\left(b_{jk}\right) = \mu_{jk} \ln\left(1 + b_{jk}\right), j = 1, 2, ...m$$

where μ_j stand for applications' coefficient. As we can see, given allocation schemes of processors and storage as mentioned above, with settled parameters such as α, β, ρ and μ_j, a bandwidth allocation scheme can be obtained, and performance metrics, e.g. data throughput and resource usage can be calculated. For bandwidth allocation, it is transformed to finding the optimal set of such parameters, which is performed at the beginning of each scheduling period, as described in Section 3.2.

Essentially, the bandwidth allocation approach proposed in our work is a type of additive increase multiplicative decrease (AIMD) algorithm that is usually used in TCP congestion avoidance. Main feature of our approach is storage awareness, since data streaming is stopped if allocated storage is nearly full such that data overflow is avoid. The approach is also processing-aware, because the processing capacity can be reflected via storage usage. Our bandwidth allocation leads to on-demand data streaming, which is also congestion aware.

Bandwidth allocation is finally implemented in our system using Globus GridFTP [8] through controlling start and end times of data transfers and tuning parallelism for each application. System implementation and experimental results are given below.

4. System Implementation and Performance Evaluation

In order to verify our approach proposed above, a grid environment is established at Tsinghua University (Beijing, China). The Globus Toolkit 4.0.1 is deployed to provide common grid services and a simple Certificate Authority is setup to sign certificates for hosts and users which are used to establish a secure and transparent environment for data streaming applications. Detailed information on our system implementation and experimental results on our resource scheduling approach is included in this section.

4.1. System Implementation

Key components in the architecture of our resource management and scheduling system include a client tool for job submission, a management engine for admission control, a scheduler, a dispatcher and application wrappers.

- Client Tool

This tool is an interface for users to submit their applications with requirement specification in XML format, including the executable, processor number and architecture, minimum bandwidth and storage requirements, data sources, etc. It is also capable of monitoring status of submitted applications and resources, halting and canceling submitted jobs.

- Management Engine

The management engine accepts users' submissions and put them into the queue, which can be accessed by the scheduler. Its main function is to provide grid supports for streaming applications, such as security, resource discovery and management. The components of Globus toolkit used here include GSI (Globus Security Infrastructure) and MDS (Monitoring and Discovery Service). The management engine is responsible for admission control.

- Scheduler

This is the core component in the whole architecture and its key algorithms are discussed in details in Section 3. It is responsible to generate scheduling parameters. Its instruction will be executed by the dispatcher.

- Dispatcher

The dispatcher is in charge of resource allocation, sending executables with their description files to appropriate processors and invoking. This component interacts with services provided by grid middleware, such as Globus GRAM.

- Application Wrappers

This component parses the description file according to the XML schemas, initializes execution of executables, and starts data streaming to specified storage with allocated bandwidth. Also, results are sent back through the dispatcher.

Besides allocation of computational resources as most traditional grid resource management and scheduling systems do, our system also deals with allocation of bandwidth and storage to support real-time data transfer, which is required by data streaming applications. Management and scheduling of processors, bandwidth and storage are carried out in an integrated rather than independent way.

4.2. Experiment Design

As shown in Figure 2, a case study for LIGO data analysis is taken as a typical data streaming application. Two streams of data from two observatories (H1 and L1), represented using two file lists, are inputs of the program, *rmon*. The program calculates the *r*-Statistic of two data sets to measure signal similarity. A signal has more possibility to be a gravitational wave candidate if it occurs at two observatories simultaneously. Once the program, *rmon*, is launched, an option file is used to define data stride and channels for each step of calculation. Data can be used in a streaming way since data are organized in time series and become obsolete after being processed. Note that this is a simplified example of LIGO data analysis since in a real world scenario there are also pre- and post-steps and a complete data analysis is usually carried out in a formal of pipelines.

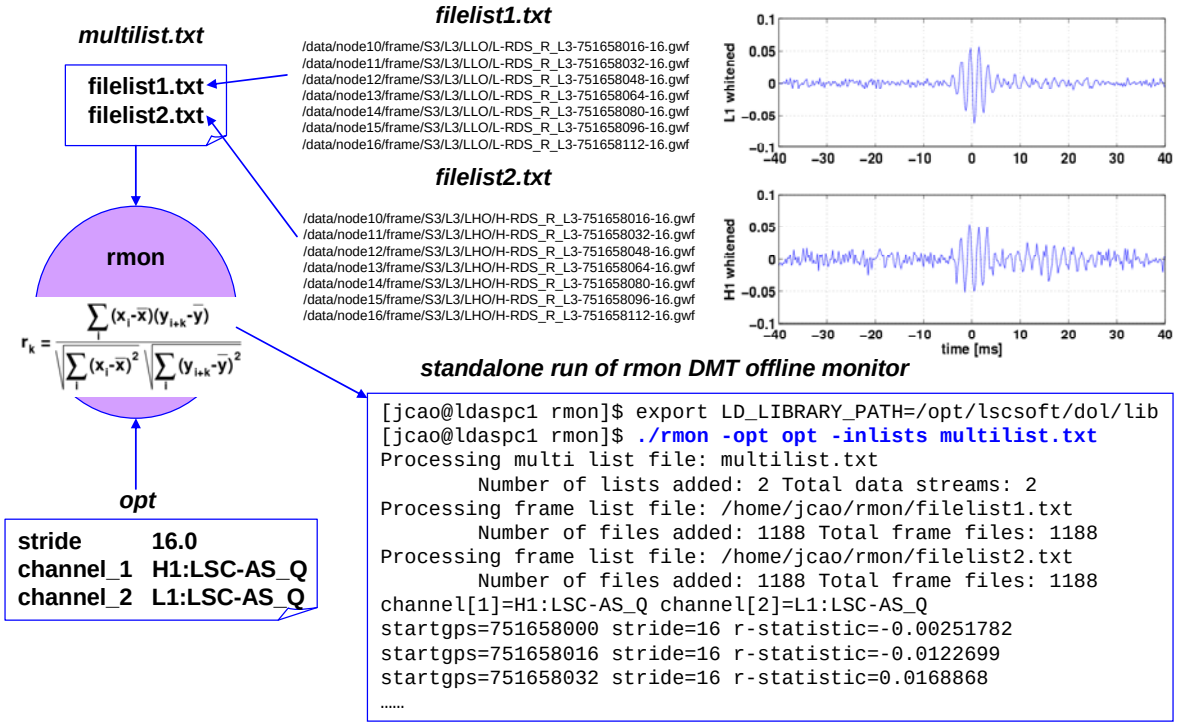


Figure 2. Experiment Design - LIGO Data Analysis Case Study

The grid environment established on campus is connected to Internet with limited bandwidth, and a network file system (NFS) is configured to which all data streams are directed. Each application is given a unique number to identify its directory in the NFS where its data is streamed so that applications can access data as if data were always available locally. In a local area network of the same campus, NSF does not introduce heavy overhead for data access. Tasks are submitted at moments complying with negative exponential distribution law, and their requirements of resources are also explicitly expressed. Experiments are carried out using parameter values listed in Table 1.

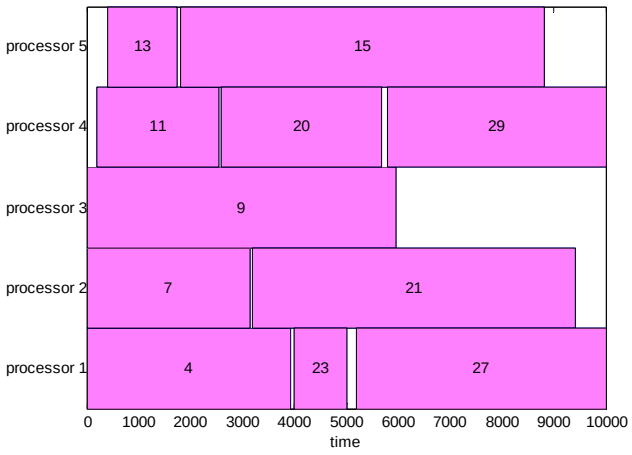
Table 1. Experiment Design - Parameter Values

Parameters	Values
m	30
n	16(5+4+7)
s	140MB
b	30Mbps
n^{req}_j, j=1,2,……,m	1
s^{req}_j, j=1,2,……,m	8 9 40 10 8 5 5 6 10 6 9 6 10 7 6 6 8 7 7 9 8 8 10 6 9 9 7 8 5 5
b^{req}_j, j=1,2,……,m	1 3 2 1 1 2 3 3 1 3 3 2 3 1 3 3 3 3 2 1 3 3 3 3 1 2 2 1 3 1
b^{max}_j, j=1,2,……,m	7 7 6 5 4 6 6 5 7 7 6 4 4 5 7 4 6 7 5 5 5 7 6 4 5 5 6 6 7 4
μ_j, j=1,2,……,m	15
l	10000
t	1s
λ	0.8

Totally 30 applications are submitted to a grid node with 16 processors, 140MB disk space, and 30Mbps bandwidth. Processors are divided into 3 groups, each with a different architecture. Each application requires one processor with different storage and bandwidth requirements. The experiment last for 10000 seconds with 1 second per time unit. Resource scheduling is carried out once per 200 time units with updated resource and application information.

4.3. Experimental Results

Figures 3-5 illustrate detailed resource usage information of the experiment. Since processors are divided into 3 groups. Each group has a separate resource scheduling scenario. While utilization of processors cannot be 100% since processors can become idle if no data are available for processing, these idle CPU time cannot be reused by other applications once allocated. Different from processor usage, storage and bandwidth usage can adapt to application requirements quickly.



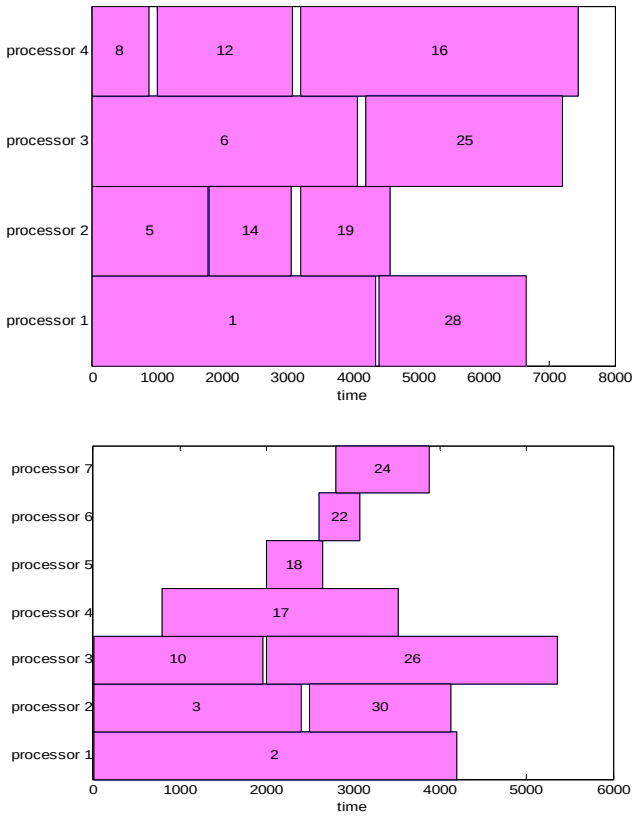


Figure 3. Experiment Results – Processor Usage

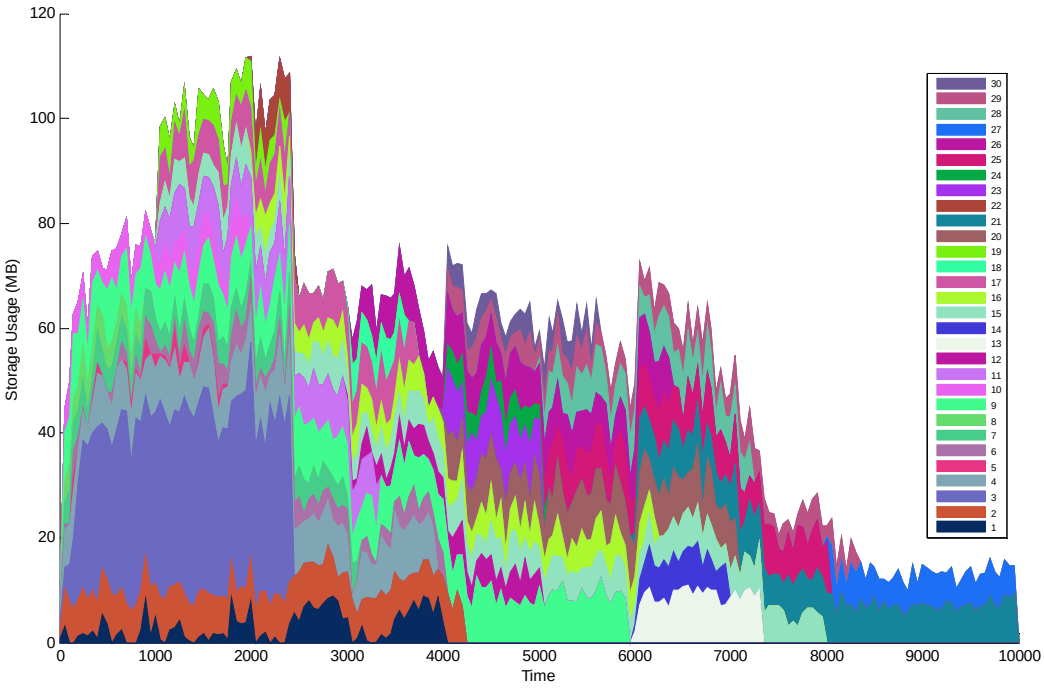


Figure 4. Experiment Results – Storage Usage

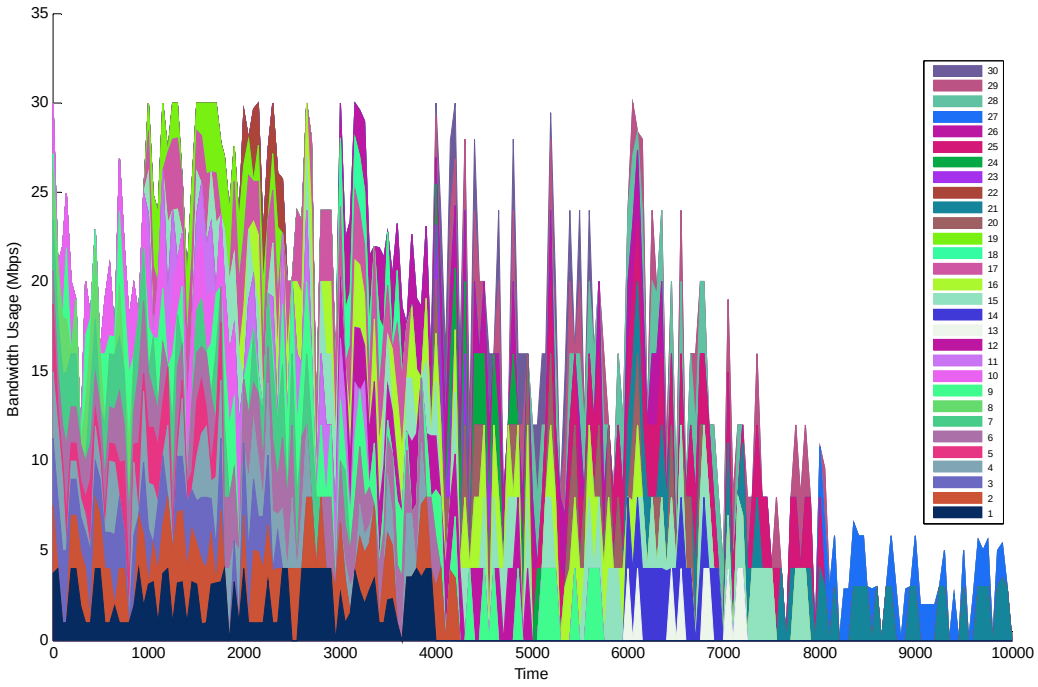


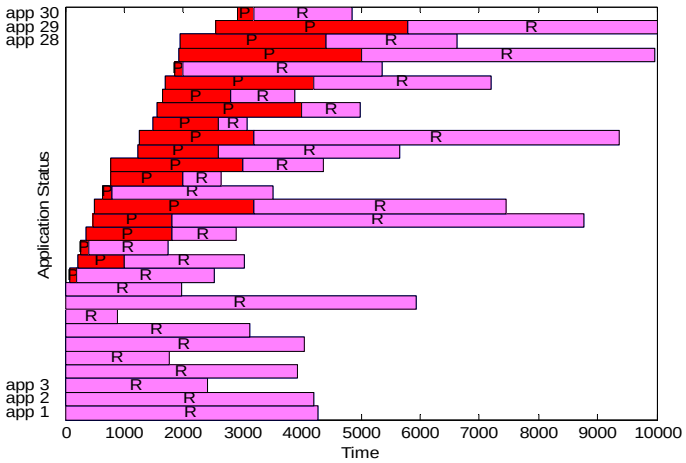
Figure 5. Experiment Results – Bandwidth Usage

4.4. Performance Evaluation

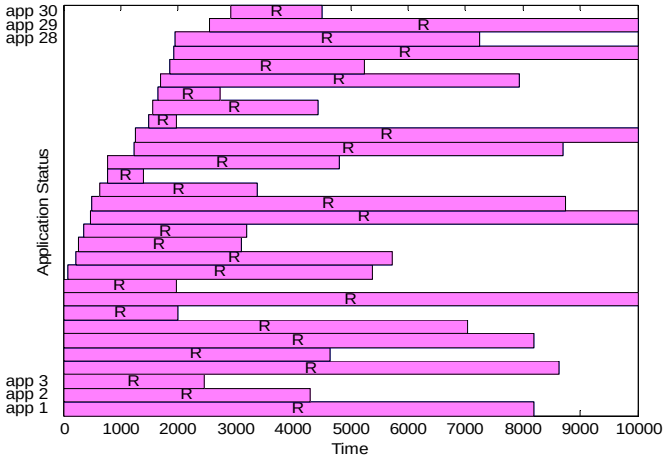
In the last section, raw data on resource usage of processors, storage and bandwidth are visualized. In this section, our approach described in Section 3 is evaluated using performance metrics described in Section 2.2.

4.4.1. Admission Control

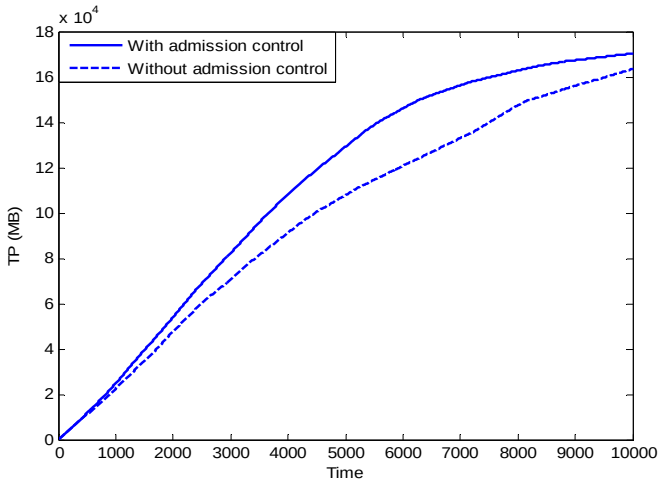
In our experiment, as grid resources are limited compared with data streaming application requirements, it can be inferred that too many applications accepted by the grid without admission control would lead to low processing performance, measured using data throughput in our case. This is verified by further experimental results included in Figure 6.



(a) With Admission Control



(b) Without Admission Control



(c) Comparison of Data Throughput

Figure 6. Performance Evaluation – Admission Control

As shown in Figure 6(c), higher data throughput is achieved in the scenario with admission control than that without admission control. Note that in the latter scenario, since no admission control mechanism is applied, all applications are started to run immediately once submitted and one processor may have to deal with more than one application at the same time. Inadequate data provision for each application and competition of computational power among applications lead to a lower data processing performance as a whole.

Figures 6(a) and 6(b) also provide an overall picture of application status in both scenarios, where red bars with the character *P* stand for pending applications and pink bars with the character *R* mean the corresponding application is running. The makespan of 30 applications is different in two scenarios. The numbers of completed applications are 29 and 25, respectively. It is obvious that the total makespan is longer in the situation when no admission control is applied.

4.4.2. Processor Utilization

As shown in Figure 3, processors are allocated to applications, one for a single application exclusively. Groups 1 and 2 deal with more applications while they include less processors than Group 3, so average workload of processors are higher than that of Group 3. Because some applications may not be able to be executed on the processors in Group 3, they cannot be transferred to Group 3 for load balancing.

Once an application is started to run on a certain processor, it does not mean that the processor is busy all the time. Since all applications are dependent on data streaming supports, the processor of an application could be idle if no data is available at local storage. Higher processor utilization is always required for a better system performance.

In this section, an additional experiment is carried out with a scheduling scheme that allocation of computing, bandwidth and storage resources is made independently. As shown in Figure 7, compared with the integrated approach proposed in this work, the independent scheme results in lower processor utilization for most of applications. Higher processor utilization is achieved using our integrated scheduling scheme.

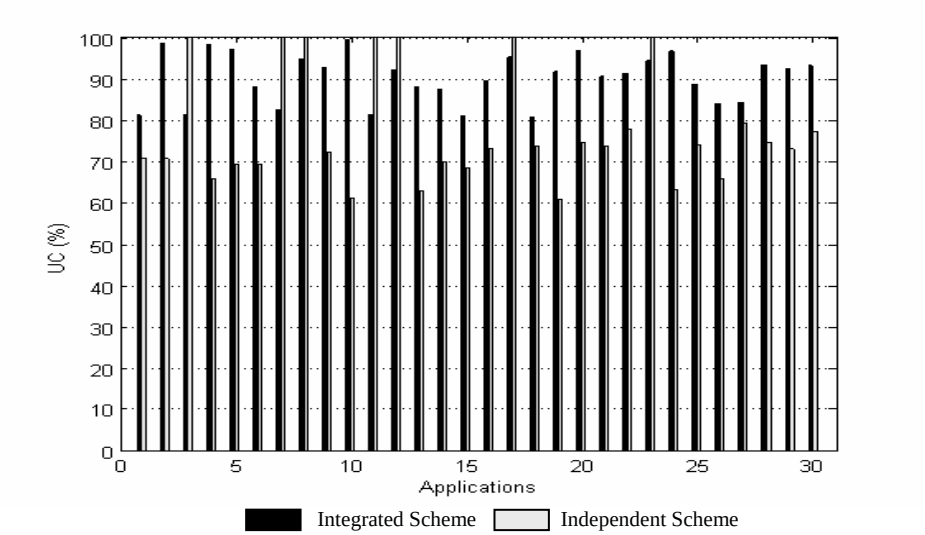
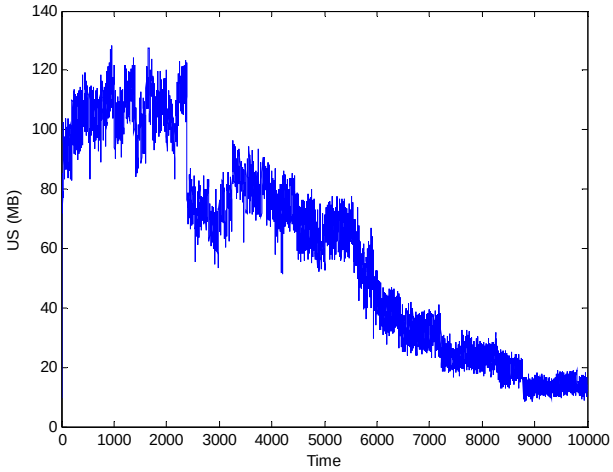


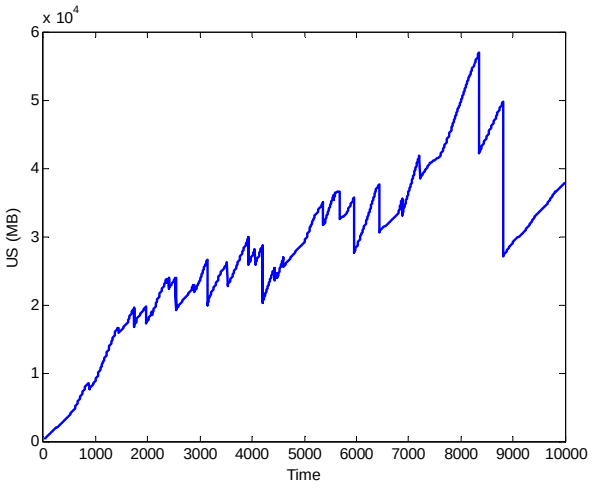
Figure 7. Performance Evaluation – Processor Utilization

4.4.3. Storage Awareness

Another feature of our approach is that data streaming is storage-aware, i.e., data transfers are controlled by the usage of allocated storage, rather than spontaneously. The principle here is that just-enough-data is ok, not the-more-data-the-better. Data streaming can be intermittent, not always continuous. In this way high volume of data can be processed with reasonable storage usage, as shown in Figure 8(a). Storage usage varies in a reasonable scope during the experiment. If data streaming is continuous and available storage space is large enough, storage usage can be of high volume, as illustrated in Figure 8(b). Small storage can achieve high throughput for data streaming applications with well-controlled data streaming and processing scheme.



(a) Storage-Aware Data Streaming



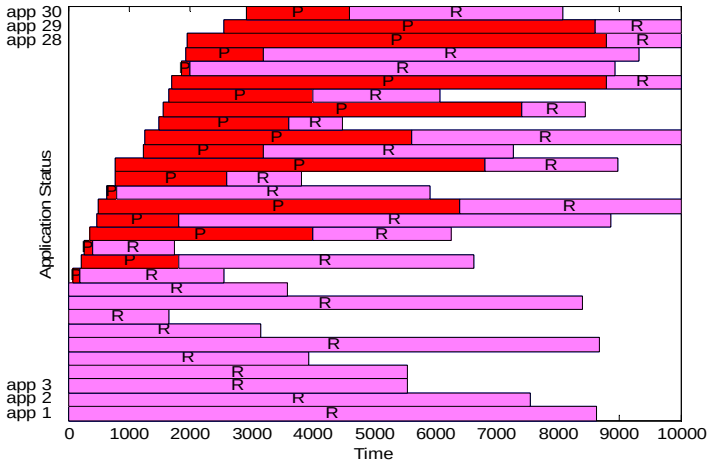
(b) Non-Storage-Aware Data Streaming

Figure 8. Performance Evaluation – Storage Aware Data Streaming

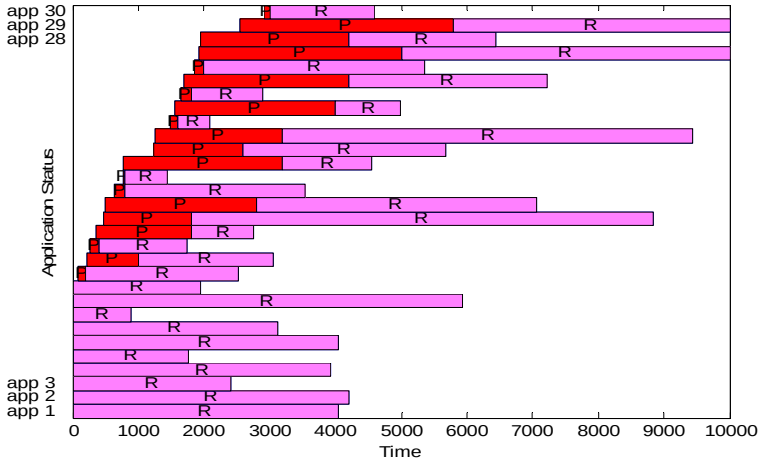
4.4.4. Iterative Bandwidth Allocation

Bandwidth is allocated to each running application to guarantee their data provision. Parameters for bandwidth allocation are obtained using the GA (as described in Section 3.2) and applied in each scheduling period. As described in Section 3.3.3, bandwidth allocation is an iterative process that is adaptive to the total available bandwidth and requirements of running applications.

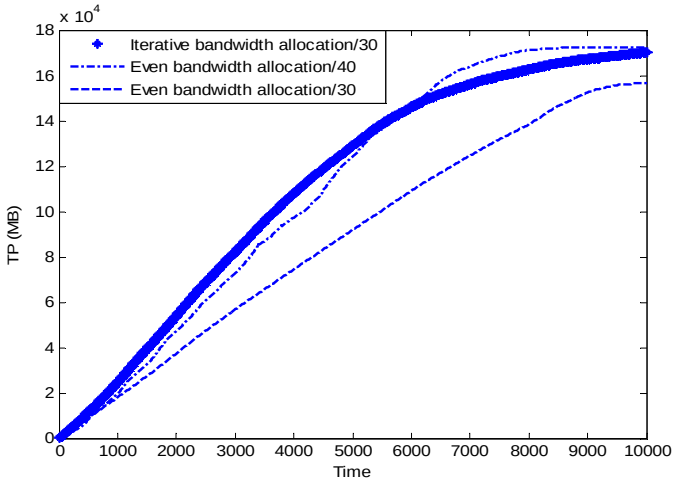
To justify our iterative bandwidth allocation algorithm, additional experiments are carried out using an even bandwidth allocation method, where bandwidth is allocated to the running applications equally. As shown in Figure 9(a), since total available bandwidth is relatively low (b=30Mbps), with even bandwidth allocation, only 25 applications are finished in 10000 time units. In Figure 9(b), available bandwidth is increased to 40Mbps. In this case, each application can get enough data with the even allocation scheme. These results can be compared with Figure 6(a). Using our iterative approach, a better performance can be achieved with relatively low available bandwidth.



(a) With Even Bandwidth Allocation (b=30)



(b) With Even Bandwidth Allocation (b=40)



(c) Comparison of Data Throughput

Figure 9. Performance Evaluation – Bandwidth Allocation

Comparison of data throughput using iterative and even bandwidth allocation is illustrated in Figure 9(c). It is obvious that higher data throughput can be achieved using our iterative bandwidth allocation, though only relatively low available bandwidth is available. This is because that our approach is storage- and processing-aware. Using the even allocation method, some applications may starve for data while others may be allocated redundant bandwidth, which reduces data processing efficiency and results in lower data throughput.

5. Related Work

Stream processing used to be the focus of database research [9][10][11][12] in recent years, and some tools and techniques have been developed to cope with efficient handling of continuous queries on data streams, while our work focus on scheduling streaming applications on grid resources.

Grid scheduling has primarily focused on providing supports for batch-oriented jobs [13]. Most resource management infrastructures available for grid computing, such as Legion [14], Nimrod/G [15], Condor [16] and ARMS [17], are largely geared to support batch-oriented applications rather than streaming ones. Attention has also been paid to scheduling of interactive job in grid environments [18][19][20]. There are several existing efforts that are focused on grid resource management and scheduling for data streaming applications.

- GATE

A middleware system, called GATES [21][22][23] (Grid-based AdapTive Execution on Streams), is developed for those applications involving high-volume data streams and requiring distributed processing of data arising from a distributed set of sources. A resource allocation algorithm based on minimal spanning tree (MST) is developed, whose target is to create a deployment configuration, including (1) the number of data sources and their location; (2) the destination, i.e. the node where the final results are needed; (3) the number of stages in the application; (4) the number of instances of each stage; (5) how the instances connect to each other and (6) the node at which each stage is assigned. Given a streaming application composed of several stages, the first three components of a deployment configuration are determined, and the emphasis of this resource allocation algorithm is to decide the last three components to give the applications the best chance to achieve best performance. Once a deployment configuration is finished, the launcher program can be called to automatically launch the application.

- Streamline

Streamline [24][25], as a scheduling heuristic, is designed specially to adapt to the dynamic nature of grid environment and varying demands of a streaming application. It expects to maximize throughput of the application by assigning best resources to the neediest stage in terms of computation and communication requirements, belonging to a general class of list scheduling algorithms. Stages are prioritized according to their estimated computation and communication costs, and the resources leading to minimal costs will be assigned to the stages. In this algorithm, the precise estimation of computation and communication costs must be provided, which is not easy to be implemented.

- Pegasus

Pegasus [26][27] is aimed to address the problem of automatically generating job workflows for the grid, which helps map an abstract workflow defined in terms of application-level components to the set of available grid resources. It handles data transfers, job processing and data cleanups in a workflow manner, not in an integrated and cooperative way as proposed in our approach.

The above projects mainly focus on management and scheduling of computational and data resources for grid applications, where network resources are not considered. The following two projects provide co-allocation of various grid resources but they are not targeting data streaming applications.

- EnLIGHTened

As best-effort networks always introduce bottleneck for high-end applications, EnLIGHTened computing [28] project is aimed to co-allocate any type of Grid resource: computers, storage, instruments, and deterministic, high-bandwidth network paths, including lightpaths, in advance or on-demand. Then external networks, especially dedicated optical networks, can be used as first class resources within gird environments, which will guarantee large data transfers associated with executions on remote resources. Its idea is similar with our integrated resource scheduling proposed in this paper, but we implement more fine-grained on resource allocation to cope with specific requirements for data streaming applications, e.g., storage awareness.

- G-lambda

G-lambda [29] project is to define a standard web service interface (GNS-WSI) between a grid resource manager and a network resource service from a network resource manager provided by commercial network operators. One grid scheduling system is developed to co-allocate computing and network resources with advance reservations through web service interfaces using the Grid Resource Scheduler (GRS), the Network Resource Management System (NRM), which is capable of GMPLS network resource management. This is a general framework for resource co-allocation, which does not pay enough attention to characteristics of data streaming applications, such as sustaining and controlled data provision.

6. Conclusions and Future Work

Data streaming applications bring new challenges to grid resource management and scheduling, such as requiring real-time data provision and integrated resource allocation schemes. Different from existing resource management and scheduling schemes that only focus on computational resources, the system proposed in this paper takes computational, storage and network resources into account simultaneously and make integrated management and scheduling schemes, which are proved to be feasible with improved performance and scalability.

The work described in this paper is mainly focused on resource utilization and overall data throughput of the system. Future work will address quality of service (QoS) issues required by data streaming applications. Scheduling for data streaming pipelines will also be considered, which is more complicated with requirement of balancing among multiple stages and appropriate data provision. Ongoing work also includes scheduling data sharing among multiple data streaming applications to further improve system performance.

Acknowledgement

This work is supported by National Science Foundation of China (grant No. 60803017), 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 his long-term collaboration supports.

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]. J. Cao, E. Katsavounidis, and J. Zweizig, "Grid Enabled LIGO Data Monitoring", *Proc. IEEE/ACM Supercomputing Conf.*, Seattle, WA, USA, 2005.

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

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

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

[7]. K. Kar, S. Sarkar, L. Tassiulas, "A Simple Rate Control Algorithm for Maximizing Total User Utility", *Proc. INFOCOM* 2001.

[8]. 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.

[9]. D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A New Model and Architecture for Data Stream Management", *VLDB Journal*, vol. 12, No. 2, pp. 120-139, 2003.

[10]. M. Balazinska, H. Balakrishnan, and M. Stonebraker, "Contract-based Load Management in Federated Distributed Systems", *Proc. 1st Sym. on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[11]. S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "TelegraphCQ: Continuous Dataflow Processing", *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'03)*, 2003.

[12]. M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik, "Scalable Distributed Stream Processing", *Proc. 1st Biennial Conf. on Innovative Data Systems Research (CIDR'03)*, Asilomar, CA, January 2003.

[13]. J. Nabrzyski, J. M. Schopf, and J. Weglarz, *Grid Resource Management: State of the Art and Future Trends*, Kluwer Academic Publishers, Sep 2003.

[14]. 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.

[15]. R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid", *Proc. High Performance Computing ASIA*, 2000.

[16]. 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.

[17]. J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson and G. R. Nudd, "ARMS: an Agent-based Resource Management System for Grid Computing", *Scientific Programming, Special Issue on Grid Computing*, Vol. 10, No. 2, pp. 135-148, 2002.

[18]. S. Basu, V. Talwar, B. Agarwalla, and R. Kumar, "I-GASP: Interactive Grid Environment Provided by Application Service Providers", *Proc. 1st Int. Conf. on Web Services (ICWS'03)*, Las Vegas, USA, 2003.

[19]. V. Talwar, B. Agarwalla, S. Basu, and R. Kumar, "Architecture for Resource Allocation Services Supporting Remote Desktop Sessions in Utility Grids", *Proc. 2nd Int. Workshop on Middleware for Grid Computing (MGC 2004)*, Toronto, Canada, Oct. 2004.

[20]. J. Cao and F. Zimmermann, "Queue Scheduling and Advance Reservations with COSY', *Proc. 18th IEEE Int. Parallel & Distributed Processing Symp.*, Santa Fe, NM, USA, pp. 63, 2004.

[21]. L. Chen, K. Reddy, and G. Agrawal. "GATES: A Grid Based Middleware for Processing Distributed Data Streams". *Proc. 13th IEEE Int'l. Sym. on High Performance Distributed Computing (HPDC-13)*, Honolulu, Hawaii USA, June 4-6 2004.

[22]. L. Chen and G. Agrawal, "Resource Allocation in a Middleware for Streaming Data", Proc. *2nd Workshop on Middleware for Grid Computing (MGC'04)*, Toronto, Canada, Oct 18 2004.

[23]. L. Chen and G. Agrawal, "A Static Resource Allocation Framework for Grid-based Streaming Applications", *Concurrency and Computation: Practice and Experience*, Vol. 18, pp. 653–666, 2006.
[24]. 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.
[25]. 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.
[26]. 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.
[27]. 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.
[28]. L. Battestilli, et al., "EnLIGHTened Computing: An Architecture for Co-allocating Network, Compute, and other Grid Resources for High-End Applications", *Proc. Int. Symp. on High Capacity Optical Networks and Enabling Technologies*, pp. 1-8, 2007.
[29]. A. Takefusa, et al., "G-lambda: Coordination of a Grid Scheduler and Lambda Path Service over GMPLS", *Future Generation Computer Systems,* Vol. 22, No. 8, pp. 868-875, October 2006.