An Integrated Resource Management and Scheduling System for Grid Data Streaming Applications

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

Abstract

Grid data streaming applications are novel from others in that they require real-time data supply while the processing is going on, which necessitates harmonious collaborations among processors, bandwidth and storage. Traditional scheduling approaches may not be sufficient for such applications, for they usually focus on only one aspect of resources, mainly computational resources. A resource management and scheduling system for such applications is developed in this paper, which is responsible for enabling their running based on Globus toolkit. An integrated scheme is proposed, including admission control, application selecting, processor assigning, allocation of bandwidth and storage, with corresponding algorithms elaborated. Evaluation results show excellent performance and scalability of this system.

1. Introduction

Streaming applications are gaining their popularity recently, and in most cases data are *pushed* to the computational resources for distributed processing with real-time constraint, so the processing rate must match the data arrival rate. Nowadays, new kinds of streaming applications are emerging with different requirements and characteristics. For example, LIGO (Laser Interferometer Gravitational-wave Observatory) [1] is generating 1TB scientific data per day and trying to benefit from processing capabilities provided by the Open Science Grid (OSG) [2]. 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. Such applications are novel in that (1) they are continuous and long running in nature; (2) they require efficient transmission of data from/to distributed sources/sinks in an end-user-pulling way; (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 compute-intensive tasks in a timely manner. Grid computing [3] paves a new way for such kinds of applications, giving birth to the so-called Grid Data Streaming applications.

Such applications require the combination of bandwidth sufficiency, adequate storage and processors to guarantee smooth and high-efficiency processing, making them different from other batch-oriented ones. Most scheduling infrastructures available in the filed of grid, such as Legion [4], Nimrod/G [5] and Condor [6], are largely geared to support batch-oriented applications rather than the streaming ones. Some schedulers are developed to support data streaming applications, such as E-Condor, GATES [7], and Streamline [8], but they just concern on computational resource allocation, paying little attention to storage and network bandwidth. Pegasus [9] has the most similar motivation with the work described in this paper, but it handles data transfers, job processing and data cleanups in a workflow manner. EnLIGHTened computing [10] and G-lambda [11] project, which provide co-allocated computing and network resources with advance reservation, but they don't concern with specific requirements of Grid data streaming applications.

In this paper, an integrated resource management and scheduling system is developed from viewpoint of the resources, including processor, storage and bandwidth, to make efficient use of them and accommodate as many streaming applications as possible to achieve high throughput. This resource

management and scheduling system tries to allocate processors, storage and bandwidth synchronously to guarantee such applications to execute smoothly with high efficiency. Based on Globus toolkit [12], this system is able to discover and manage resources geographically distributed and belonging to different management domains in a transparent and secure way. Some key algorithms are proposed, including admission control, application selecting, processor assigning, bandwidth allocation and storage allocation. Evaluation results show excellent performance and scalability of this system.

The rest of this paper is organized as following: Section 2 describes the overall architecture and mechanism of this resource management and scheduling system, whose core algorithms are elaborated in the next section; some evaluation results are included in Section 4, and the following section concludes this paper.

2. System Architecture

The architecture of our resource management and scheduling system is shown in Figure 1 and its key components include but are not limited to:

- Client Tool

This tool is an interface for users to submit their applications with their requirements in XML format, including the executable, processor types and amount , minimum bandwidth and storage, data source, just like but more than what Condor submission does. It is also capable of monitoring the status of submitted applications and that of the resources in the whole grid. Nowadays, it is carried out in command lines, and in the future a graphical user interface (GUI) will be available.

- Management Engine

The management engine accepts users' submissions of applications and put them into the queue, which will 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 GRAM (Globus Resource Allocation Manager), MDS (Meta-computing Directory Service), GSI (Globus Security Infrastructure), GASS (Global Access to Secondary Storage), NWS (Network Weather Service), GRIS (Grid Resource Information Service), GIIS (Grid Index Information Service) and so on.

- Scheduler

This is the core component in the whole architecture and its key algorithms will be discussed in details in Section 3. It is responsible to carry out admission control, application selecting, processor assignment, and bandwidth and storage allocation. Its instruction will be executed by the dispatcher.

- Dispatcher

The dispatcher is in charge of sending executables with their description files to appropriate processors and invoking a remote component, i.e., application wrapper. This component will interact with the services provided by grid middleware, such as GRAM.

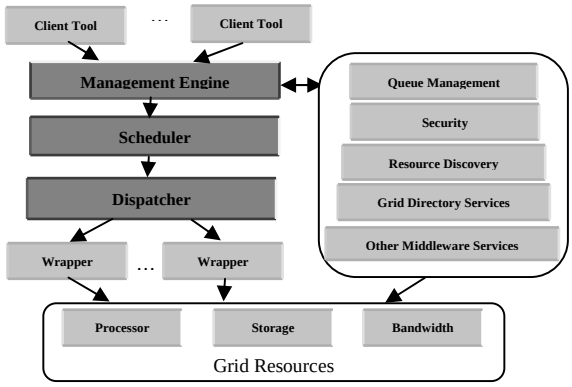


Figure 1. System architecture

- Application Wrapper

This component will parse the description file according to the XML schemas, initialize execution of executables, and start data transmission to specified storage with allocated bandwidth. Also, it will send back the results through dispatcher. Another function is to monitor the usage of storage to determine data transmission status, see more details in subsection 3.5.

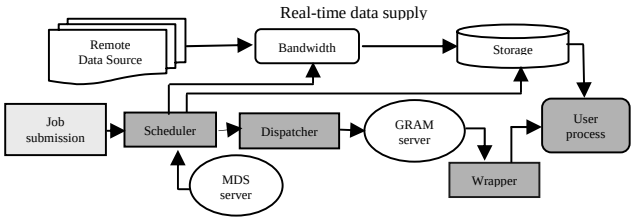


Figure 2. Overall mechanism

The overview of the running mechanism is illustrated in Figure 2. Besides allocation of computational resources as most traditional resource management and scheduling systems do, it also deals with allocation of bandwidth and storage to support real-time data supply, which is required by data streaming applications. Management and scheduling of processors, bandwidth and storage are carried out in an integrated way rather than independently.

3. Key Algorithms

This section just elaborates on the key algorithms as the core of this resource management and scheduling system, i.e., the scheduler. Note that although processor assignment, allocation schemes for storage and bandwidth are described and evaluated

separately, they are carried out synchronously as integration.

3.1. Admission control

When a new job is submitted, admission controller would decide to run it instantly or just keep it in the waiting queue. This decision is made according to the usage status of resources and the requirements of the jobs. Each job can allege its minimum requirement of resources, e.g., it needs some processors, bandwidth and storage. An XML schema is developed for the applications to express their requirements in the manner similar to Resource Description Language (RSL).

For each application *s*, it can declare its minimum requirement of resources like

$$R_s = [p_s \ b_s \ st_s]$$

where p_s stands for the number of processors it requires, so p_s=1 for simple applications (i.e., standalone applications) and p_s>1 for composed applications (such as a pipeline); b_s and st_s stand for the required minimum bandwidth and storage respectively. This information will be included in the submission file in XML format.

Suppose the running applications in the computing pool form a set, denoted as S_R, and the total amount of processors, bandwidth and storage be denoted as *P*, *B* and *S* respectively. Some applications have their special requirements upon processors, for example, applications compiled on X86_64 cannot run on I386 processors, so not every processor is suitable for each application. Suppose those processors eligible for application *s* form a set, called P_s, and the number of free (not occupied or reserved) processors in it when *s* comes is denoted as $|P_s|$.

In any one of the following three cases, a new application, s_n, would just be kept in the waiting queue for there are no enough resources (suitable and enough processors, enough bandwidth and enough storage respectively) for it.

$$p_{s_n} > \left|P_{s_n}\right|$$

$$b_{s_n} > B - \sum_{s \in S_R} b_s$$

$$st_{s_n} > S - \sum_{s \in S_R} st_s$$

If an application's minimum requirement can be satisfied according to the current status of computing pool, it will called a potential eligible application (PEA), which means that it *may* be permitted into computing pool.

3.2. Application selection

PEAs form a queue, within which maybe several ones satisfy the admission control policy. A selecting policy must be applied to choose some from the queue and assign appropriate resources for them. Those selected ones will be called eligible applications (EAs).

PEAs have different weights, and the higher weights mean that they can be selected with bigger priorities. PEAs will be classified into several groups according to their weights, and in each group, the selecting principle is first-come-first-serve (FCFS).

The selecting will be heuristic and iterative: the first coming PEA with the highest weight will be selected, and then the next one till the last one in its group (if there are) will be tested in their arriving order; then it is turn for the group with second highest weight, till all the groups are tested. Notice that the PEAs with higher weight will not be selected prior to those with lower weight necessarily, for whenever a PEA is accepted, the resource status will change and some PEAs will become ineligible.

To some extent, this algorithm resembles first-fit (FIFT) with backfilling mechanism. What is more, to avoid that some PEAs starve for a long time, some reservation policy will be adopted. Some resources will be labeled as reserved when they are executing other applications, and as soon as they are free, they will be assigned to the applications which reserve them. Weights of each application will increase as time goes by, to avoid such cases where applications with lower weights will be idle forever. The weights will be a function of time, with the originally set value as their initializations

$$w_i(t) = f_i(w_{io}, t)$$

$$f_i(w_{io}, 0) = w_{io}$$

where w_{i0} is the initial weight of application *i* and *f(t)* is an non-decreasing function about time *t*. A function in case is

$$w_i(t) = f_i(w_{io}, t) = w_{io} + d_i * f(t, T_i)$$

$$f(t, T_i) = floor(t / T_i)$$

where d_i is the increase coefficient and d_i>*0*; T_i is the increase period and function *floor* returns the nearest integer towards minus infinity for *t* divided with T_i . Then w_i will increase by d_i once a period T_i. Assigning appropriate values for d_i *and* T_i, after some time of waiting, the applications with lower weights initially will be endowed a high enough weight to be selected from PEA queues.

Combination of reservation policy and increasing weight over time will guarantee each application will be accepted by the computing pool in appropriate time.

In one word, the selecting algorithm tries to make full use of resources and keep fairness among applications.

3.3. Processor assignment

As soon as EAs are selected, it is time to assign resources for them. Applications may have their own styles, i.e., they may be executed more smoothly on some processors than on others. So it is necessary to assign appropriate processors for applications, and purely random assignment will not work.

On the other hand, the processors can be classified into several groups according to their characteristics, their architecture for instance. One application will achieve similar performance on the processors of a group, so it is not necessary to launch it on each processor for trial, but a processor can act as the representative of its peers in the same group.

Matchmaking will be carried out to find candidate processors for applications, and applications will be assigned to processors in the matched group to run a short period of time to get its performance information. The applications with higher weights will have higher priorities to find their matched processors, and the processors producing the highest processing efficiency will be selected.

3.4. Storage allocation

When new EAs arrive, the scheduler is responsible for allocating bandwidth and storage for them, together with the existing applications in the computing pool.

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 pool, the storage allocated for each application can be set to a high value. While the applications increase, the allocated storage for each application may be decreased. There must be some margin of storage for potentially coming applications. An iterative allocation algorithm of storage is proposed as following:

① initialization: suppose there are n applications in the pool, to generate n random numbers, $r_i \in (0, 1)$, i=1,2,…,n. Calculate each quota, q_i as following

$$q_i = \frac{r_i}{\sum_{j=1}^{n} r_j}$$

② If $q_i * TS \geqslant st_i$, reserve these numbers for initially allocated storage for application i; else, repeat step ① until all of these inequations hold true, where st_i is the minimal required storage of application i as mentioned in subsection 3.1 and TS is the total storage available for applications.

③ dynamic adjustment: periodically, monitoring usage status of each allocated storage, and those with high occupation percentages will be increased while others will be decreased;

④ when a new EA is coming, decrease the amount of the biggest partition of storage;

⑤ when an application is finished, its storage will be divided and allocated to the minimal partitions;

⑥ repeat ③, ④ and ⑤ until all the applications are completed.

3.5. Bandwidth allocation

Bandwidth allocation plays an important role in the whole resource allocating scheme, for appropriate bandwidth is indispensable to guarantee data supply for applications to make them run constantly. To make a flexible allocation scheme, so-called utility functions are introduced and genetic algorithm [13] is adopted to maximize their sum. Different from traditional bandwidth allocation, our scheme is *storage aware*, i.e., data transmission 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, transmission will be halted for a while until some data have been processed and cleaned up so that some storage is released for more data. At any moment, the amount of data in storage for each application is affected by data supply and clean-up at the same time, where the former tends to increase the amount while the latter will decrease it.

The computing pool is connected to Internet through which the data are streamed to the applications being executed on the processors, and the total input bandwidth, denoted as I, is limited, which is shared by the data streams. The data streams, called sessions, denoted as s, form a set S. Each session will be assigned a bandwidth x_s, where $x_s \in X_s$, $X_s = [b_s, B_s]$ and $b_s > 0$, $B_s < \infty$. b_s stands for the least bandwidth required for session s, while B_s is the highest bandwidth available for s from the corresponding data source. Session s will have a utility $U_s(x_s)$ when its data is supplied at a rate x_s, where $U_s(x_s)$ is called utility function and assumed to be concave, continuous, bounded and increasing in the interval $[b_s, B_s]$. Note that it is not necessary that all the sessions adopt identical utility functions. We try to maximize the sum of the utilities of all the sessions, maintaining fairness among them. The problem can be described as follows.

P.

$$\max \sum_{s \in S} U_s(x_s) \tag{1}$$

s.t.

$$\sum_{s \in S} x_s \leq I \tag{2}$$

$$x_s \in X_s \tag{3}$$

Due to the usage status of storage, there are two possible states for each s at any time, i.e., active and inactive, which indicate a data transmission is on or off. All the active sessions form a set, called S_A, and it is obvious that this set is varying because the states of sessions are changing.

We just allocate bandwidth for active transmissions, so the constraint (2) may be rewritten as

$$\sum_{s\in S_A} x_s \le I \tag{4}$$

An iterative optimization algorithm is proposed in [14] and its convergence is analyzed, but it is required to be aware of the congestion on the path, which is hard to be satisfied in the wide Internet. According to our situation, we make some modification upon it as following.

While $s \in S_A$

$$x_s^{(k+1)} = \begin{cases} \left[x_s^{(k)} + \alpha_k U'\left(x_s^{(k)}\right)\right]_{X_s} & if \sum_{s\in S_A} x_s^{(k)} \le \rho I \\ \left[\beta_k x_s^{(k)}\right]_{X_s} & if \sum_{s\in S_A} x_s^{(k)} > \rho I \end{cases} \tag{5}$$

Otherwise

$$x_s^{(k+1)} = 0, \forall s \notin S_A \tag{6}$$

Here, $x_s^{(k)}$is the bandwidth for session $s\in S$ at the k^{th} step. $\{\alpha_k\}$ and $\{\beta_k\}$are two positive sequences. For the sake of convenience, α_k and β_k are usually substituted as a fixed value, denoted as α and β respectively. $[\cdot]_{Xs}$ denotes a projection on the set X_s and can be calculated as

$$[y]_{X_s} = \min(B_s, \max(b_s, y))$$

$U'(\cdot)$ is the sub gradient of

$$U(\cdot) = \sum_{s\in S_A} U_s\left(x_s^{(k)}\right) \tag{7}$$

and

$$U'\left(x_s^{(k)}\right) = \frac{\partial U(\cdot)}{\partial x_s^{(k)}}$$

If we define

$$U_s(x_s) = 0, \forall s \notin S_A$$

formula (7) can be modified as

$$U(\cdot) = \sum_{s\in S} U_s(x_s)$$

And ρ is the so-called safety coefficient to avoid bandwidth excess, where $\rho\in$ (0, 1), i.e., there is some margin from the full use of total bandwidth for flexibility and robustness. Some heuristic algorithms, such as genetic algorithm will be applied to find optimal values for them.

A popular utility function can be expressed as

$$U_s(x_s) = \mu_s \ln(1+x_s),\ s \in S$$

where μ_s may stand for the session's coefficient.

Essentially, bandwidth allocation here is a kind of additive increase multiplicative decrease (AIMD) algorithm which is usually used in TCP congestion avoidance, but it has some pleasant new characters. It is *storage-aware*, for the transmission will be stopped if allocated storage is nearly full, so data overflow will be avoided while enough data are supplied; it is *processing-aware*, for the processing capacity of processors will be inflected in the varying occupation of storage, although it may not aware of the precise values, and such bandwidth allocation is on-demand; of course, it is *congestion aware*.

As mentioned above, the scheduler should make allocation schemes in an evolving way to keep pace with latest situation, i.e., when an application is submitted or finished, or resources increase or decrease dramatically, it will be invoked to make new schemes to correspond to latest conditions.

4. 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. This campus grid is connected to Internet with limited bandwidth, and network file system (NFS) is established to which all the data streams are directed.

Applications are submitted at moments complying with negative exponential distribution law, and their requirements of resources are also explicitly expressed. Experiments are carried out for 10,000 units of time and some results are obtained. Resource scheduling is carried out once per 200 units of time to correspond to updated situations.

4.1. Admission control

As the resources in the computing grid are limited and each streaming application holds its own requirement, it can be inferred that too many applications accepted by the computing grid will lead to low processing efficiency if no admission control is carried out, and some experimental results verify it, as shown in Figure 3.

Higher throughput is achieved in the scenario with admission control than that in the scenario without admission control. Note that in the latter scenario, one processor may have to deal with more than one

application at the same time, which offends the assumption made before that one application will occupy a processor exclusively. Inadequate data supply for each application and discount of computational capacity due to competition among applications on one single processor lead to a lower data processing efficiency as a whole.

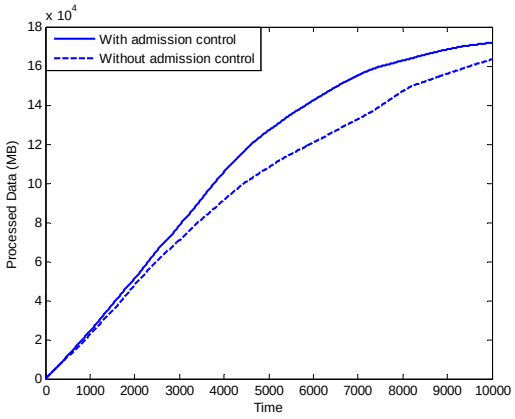
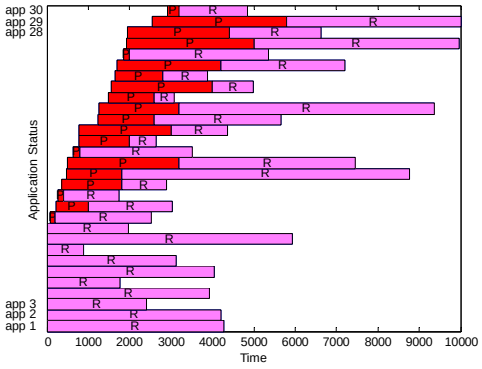


Figure 3. Throughput with/without admission control

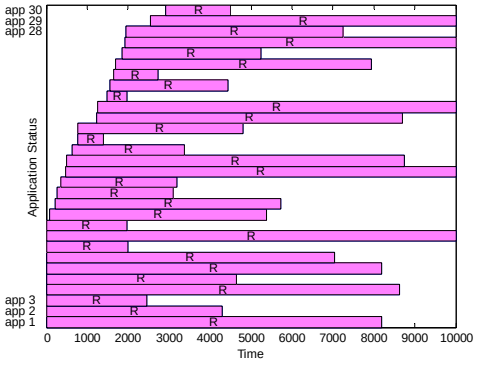
What's more, admission control can make applications finished sooner than without, as demonstrated in (a) and (b) in Figure 4, where the red bars with character P stand for the pending status and pink bars with character R mean for the running status for each application. The numbers of completed applications are 29 and 25 respectively. More importantly, most of the makespans without admission control are longer than their counterparts, which is adverse for the requirements of quality of service.

4.2. Bandwidth allocation

Bandwidth is allocated to each running application to guarantee their data supply. Parameters in bandwidth allocation are obtained with genetic algorithm and are applied in each period. This bandwidth allocation is adaptive to the total available bandwidth and requirements of running applications.



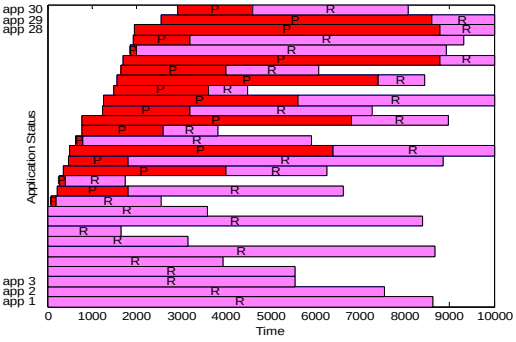
(a) With admission control



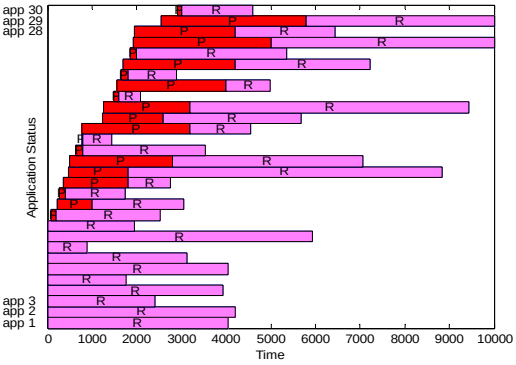
(b) Without admission control

Figure 4. Status of applications in iterative bandwidth allocation

To justify our bandwidth allocation algorithm (named iterative allocation), we compare it with the even bandwidth allocation, where the bandwidth is allocated to the running applications equably as shown in Figure 5. In (a), the total available bandwidth is relatively small which equals with that in case of Figure 3, and 25 applications are finished; in (b), the available bandwidth is relative big, so each application can get enough data supply with the even allocation scheme, and the result resembles that in (a) of Figure 4.



(a) Low bandwidth



(b) High bandwidth

Figure 5. Status of applications in even bandwidth allocation

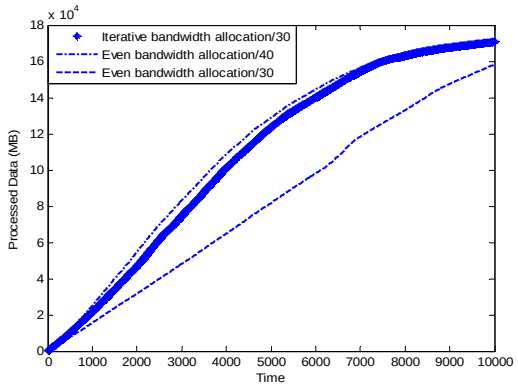


Figure 6. Throughput for bandwidth allocation schemes

Throughputs in the cases of iterative allocation and even allocation are shown in Figure 6, where the numbers at the end of legends stand for the total available bandwidth. Then our allocation scheme is justified that it can achieve high throughput with relatively small available bandwidth for it is processing-aware while the even allocation scheme is not, so in that case some applications may starve while others may be allocated redundant bandwidth.

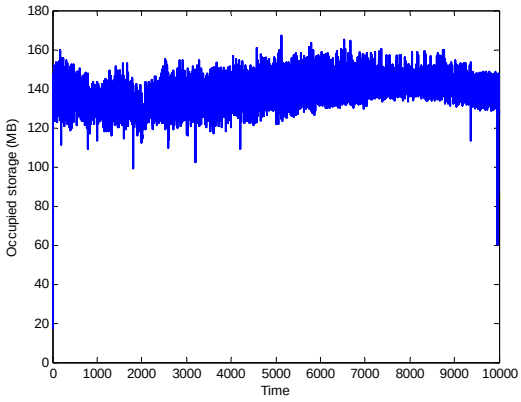
4.3. Storage usage

Data supply in our scheme is storage-aware, i.e., data supply is controlled by the usage of allocated storage, rather than spontaneously. The principle here is just enough data is ok, not the more data the better. Sometimes the data transmission is intermittent, not always continuous. In this way high volume of data can be processed with just reasonable storage, as shown in (a) of Figure 7, where the used storage just varies in a limited scope. If data supply is continuous and available storage is big enough, the occupied storage will be of high volume, which can be observed in (b) of Figure 7.

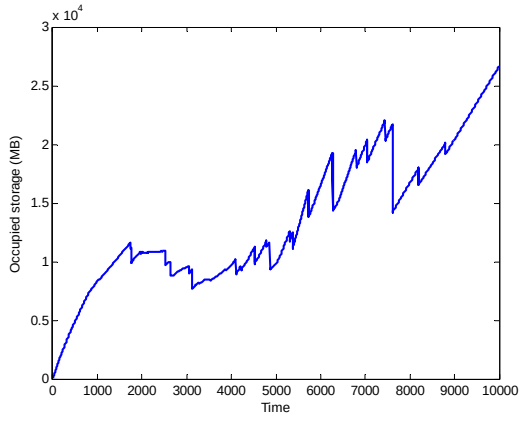
Actually, small storage can achieve high throughput in the streaming applications with well-made data supply and processing scheme, which is the prominent characteristic of such scenarios. Relative big storage is not necessary but rather desirable, for more data can be stored before processed to survive network collapse when no more data can be supplied.

4.4. Processor assignment

Processors are assigned to applications, one for a single application exclusively. Here, the allocation resembles job shop problem scheduling, as shown in Figure 8, where the numbers in the horizontal bars stand for the corresponding applications executed on the processors in a certain group. Group 1 and 2 deal with more applications while they hold less processors than group 3, so the average loads of processors are heavier that those of group 3, as demonstrated in (a), (b) and (c) respectively. Because some applications may not be able to be executed on the processors in group 3, they cannot be transferred to the processors in group 1 and 2 to make a load balance.



(a) Storage-aware data supply



(b) Non-storage-aware data supply

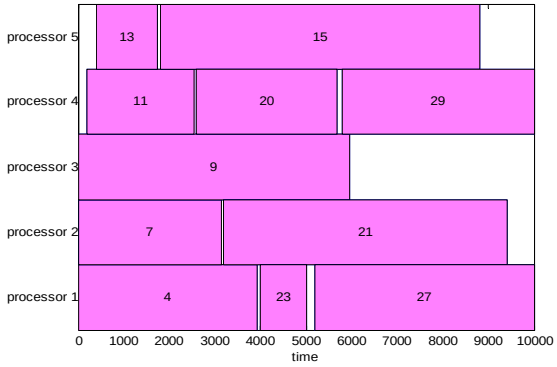
Figure 7. Storage usage

5. Conclusions

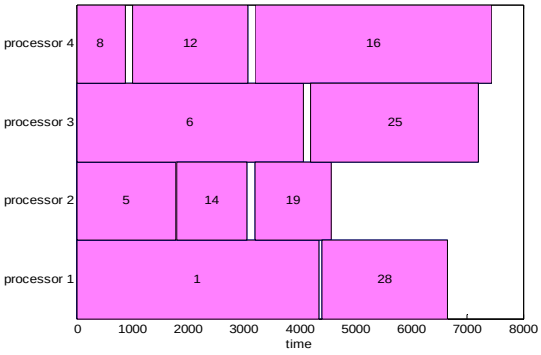
Data streaming applications are of the novel types of grid scenarios for own their characteristics, such as requiring of real-time data supply and integrated resource allocation schemes. Different from existing resource management and scheduling schemes that just focus on computing resources, the system proposed in this paper takes computational resources, bandwidth and storage into account simultaneously and make integrated management and scheduling schemes, which are proved to be feasible with excellent performance.

Up to now, requirements of quality of service (QoS) for applications have not been paid enough attention, and this desirable character will be the emphasis of further research. Scheduling for pipelined applications will be studied which is more complex with requirement of balance among stages and appropriate data supply. Ongoing work includes the consideration of data sharing scenarios among multiple data

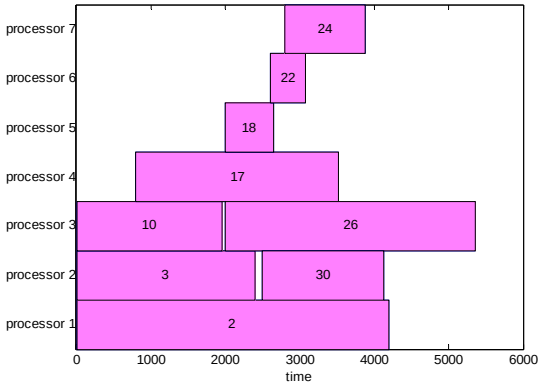
processing applications. Also some heuristic scheduling algorithm is under development for refined performance optimization.



(a) Group 1



(b) Group 2



(c) Group 3

Figure 8. Processor assignment for jobs

Acknowledgement

This work is supported by Ministry of Education of China under the higher education quality engineering project "National Open Course Integrated Systems", and Ministry of Science and Technology of China under the national 863 high-tech R&D program (grants No. 2006AA10Z237, No. 2007AA01Z179 and No. 2008AA01Z118).

References

[1]. 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.

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

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

[4]. 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.

[5]. 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.

[6]. 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.

[7]. L. Chen and G. Agrawal, "A Static Resource Allocation Framework for Grid-based Streaming Applications", *Concurrency and Computation: Practice and Experience*, 18:653–666, 2006.

[8]. B. Agarwalla, N. Ahmed, D. Hilley, and U. Ramachandran, "Streamline: a Scheduling Heuristic for Streaming Applications on the Grid", *Proc.13th Annual Multimedia Computing and Networking Conf.*, 2006.

[9]. 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.

[10]. L. Battestilli, et al., "EnLIGHTened Computing: An Architecture for Co-allocating Network, Compute, and other Grid Resources for High-End Applications", *Proc. HONET2007*.

[11]. A. Takefusa, et al., "G-lambda: coordination of a grid scheduler and lambda path service over GMPLS", *Proc. iGrid2005*.

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

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

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