Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Application-adaptive resource scheduling in a computational grid*

LUAN Cui-ju†[1,2], SONG Guang-hua[1], ZHENG Yao[1]

([1]*School of Computer Science and Center for Engineering and Scientific Computation, Zhejiang University, Hangzhou 310027, China*)

([2]*College of Information Engineering, Shanghai Maritime University, Shanghai 200135, China*)

†E-mail: cuijuluan@zju.edu.cn

Received Oct. 11, 2005; revision accepted Mar. 6, 2006

Abstract: Selecting appropriate resources for running a job efficiently is one of the common objectives in a computational grid. Resource scheduling should consider the specific characteristics of the application, and decide the metrics to be used accordingly. This paper presents a distributed resource scheduling framework mainly consisting of a job scheduler and a local scheduler. In order to meet the requirements of different applications, we adopt HGSA, a Heuristic-based Greedy Scheduling Algorithm, to schedule jobs in the grid, where the heuristic knowledge is the metric weights of the computing resources and the metric workload impact factors. The metric weight is used to control the effect of the metric on the application. For different applications, only metric weights and the metric workload impact factors need to be changed, while the scheduling algorithm remains the same. Experimental results are presented to demonstrate the adaptability of the HGSA.

Key words: Grid, Resource scheduling, Heuristic knowledge, Greedy scheduling algorithm
doi:10.1631/jzus.2006.A1634 **Document code:** A **CLC number:** TP393

INTRODUCTION

The computational grid (Foster and Kesselman, 1998) combines computing, network and storage resources to support the running of large-scale applications, which may be computation-intensive or communication-intensive. There are many technical challenges for deploying large-scale applications over this distributed computing environment due to the vast diversity of resources involved. Efficient and application-adaptive resource management and scheduling are challenging tasks in the grid. In order to meet the requirements of the applications, the scheduling algorithm must take the target applications into account. However, different applications have different characteristics, and their demands on the resources may differ greatly. The goal of our resource scheduling system is to provide a general-purpose, application-adaptive and easy-to-use scheduling approach.

We present here a Heuristic-based Greedy Scheduling Algorithm (HGSA), which aims to be adaptive to different applications. In order to rank the resources reasonably, the algorithm considers multiple resource metrics involved. Since the effect of a specific metric on different applications varies, we introduce a weight to each metric to reflect its effect on the resource scheduling. The basic idea is quite straightforward: we seek to assign customized weights to all the metrics concerned from the viewpoint of the application, consequently to select the right resources to implement the application. Moreover, we use an impact factor to identify the impact of the assigned workload of the job on the metrics. It is noted that different metrics have different impacts on the same application, and the same metric has different impacts on different applications. In order to get the suitable resources to perform the application, the user must comprehend the application-dependant characteristics and customize the weights and impact

* Project supported by the National Natural Science Foundation of China (No. 60225009), and the National Science Fund for Distinguished Young Scholars, China

factors of the metrics accordingly.

Before detailing the scheduling algorithm, we will present a distributed resource scheduling framework. On every grid site, there exist two scheduling components, i.e. global and local components. The global scheduling component uses the scheduling algorithm presented in this paper to allocate the resources. It considers all the available resources to select the appropriate ones. The local one is responsible for determining the order, in which jobs are executed at that particular site. Jobs can be scheduled at any site.

The remaining part of this paper is organized as follows. Section 2 addresses related work on resource scheduling. Section 3 describes a resource scheduling framework, while Section 4 presents the HGSA. In Section 5 experimental results are presented to illustrate its validity, and finally we summarize our work in Section 6.

RELATED WORK

The concept of a universal scheduling paradigm for any application is intractable, but some researchers have done their best to realize this task. The Condor is one of the systems, which can be used for varied applications. Liu *et al.*(2002) presented a general-purpose resource selection framework that defines a resource selection service for locating grid resources that meet the requirements of the applications. But it requires the user to provide application-specific mapping modules and ranking mechanism to personalize the resource selector, which is beyond the reach of common users.

Some resource scheduling policies have been proposed to reduce application execution time in heterogeneous environments (Chapin and Spafford, 1994; Ranganathan and Foster, 2002; Yang *et al*., 2003; YarKhan and Dongarra, 2002). However, their works only involve one or a few metrics such as network bandwidth or CPU speed, and the formula of the algorithm is fixed. But in fact, the effect of the same algorithm varies with the applications. Some scheduling algorithms are only fit for solving one type of applications.

Casanova *et al.*(2000) proposed a more sophisticated adaptive scheduling algorithm that can automatically perform on-the-fly resource selection and co-allocation of data and computation when needed. Moreover, they proposed several simple heuristics for scheduling independent tasks: Min-min, Max-min, and Sufferage, etc. But the scheduling algorithm works well with the PSAs, which focus on scheduling algorithms whose objective is to minimize the application's makespan and the metric used by the heuristics is the task's predicted completion time.

The AppLeS (Berman *et al*., 2003) project provides an environment for adaptively scheduling and deploying applications in heterogeneous, multi-user grid environments. The AppLeS can generate a schedule that considers not only predicted resource performance, but also the variation in that performance. But the applications must be customized, so that it can be dynamically scheduled by an AppLeS scheduling agent.

In order to cope with the dynamic grid environment and adapt to applications, some scheduling algorithms adopt the adaptive solution, such as (Aggarwal and Kent, 2005; Gao *et al*., 2005; Huedo *et al*., 2004; Jin *et al*., 2005). But not all of them can custom the scheduling criteria on the demand of the users.

Aggarwal and Kent (2005) also provided an adaptive generalized grid scheduling algorithm that can efficiently schedule jobs having arbitrary inter-dependency constraints and arbitrary processing durations. It is mainly used to map a set of jobs and its aim is to minimize the makespan of incoming jobs. While our algorithm is used to schedule single job, and the goal is determined by the users according to their applications.

Gao *et al.*(2005) put forward adaptive grid job scheduling algorithms that use the predicted completion time to schedule jobs at both system level and application level. In application-level scheduling, genetic algorithms are used to minimize the average completion time of jobs through optimal job allocation on each node. All their algorithms use the predicted completion time to schedule the jobs, which cannot meet various requirements of the users and their applications.

In our approach, all the measurable metrics that can influence the selection of the resources can be taken into account, and their weights can be customized by the user in order to make the scheduling al-

gorithm adaptive to a wide variety of applications. Moreover, the metrics used in the algorithm are extensible, that is, when a new metric that can affect the resource scheduling is available, such as cost, it can be added to the algorithm.

THE RESOURCE SCHEDULING FRAMEWORK

The resource scheduling framework is illustrated in Fig.1. It is a distributed one, where all sites can be both clients and servers. They have a uniform scheduling structure, which mainly consists of a job scheduler and a local scheduler. In order to simplify the diagram, some components are omitted in sites except Site 1.

The Job Scheduler (JS) is distributed and global. It schedules all the available resources in the computational grid and selects the best resources for the job by consulting the resources information. The JS gets the job to be executed from the Waiting Queue (WQ), and puts it into the Scheduled Queue (SQ). The JS adopts the first come first served (FCFS) algorithm and a two-phase scheduling strategy, which includes the filtering phase and the allocating phase. At the filtering phase, all the available resources will be filtered according to the requirements of the job, which are defined by the user. At the allocating phase, the JS chooses the best resources by utilizing the HGSA, which will be depicted in detail in Section 4.

The Local Scheduler (LS) is a job scheduler provided by the local host system, such as Open PBS, Condor or LSF, etc. It decides how to schedule the allocated jobs according to its local resources. Once a job is submitted to a particular site, which is determined by the JS, then it will be managed by the LS.

In the scheduling framework, an Information Service (IS) is in charge of resource discovery and resource information provision. It provides the JS with the available resources and their information.

All jobs managed in a site are saved in one of three job queues—the WQ, the SQ and the Finished Queue (FQ). The WQ keeps the jobs waiting for allocating resources, the SQ keeps the jobs that have been allocated resources, and finally the FQ keeps the jobs that have been executed successfully or failed.

The core function of the file controller in the framework is to provide the capabilities for the job management system to transfer file and folders securely, conveniently, efficiently and flexibly. The files and folders can be transferred by multi-channels or in third-party style. Partial file transfer can also be implemented. The file controller can also provide other file and folder operations, such as creating/deleting and validating files and folders locally and remotely. It is implemented on top of the GridFTP in the Globus toolkit (Foster *et al*., 2001).

Generally, job execution is performed in three steps.

(1) Creating the remote executing directory, and transferring the executable and all the files needed for remote execution, such as parameter files and data files.

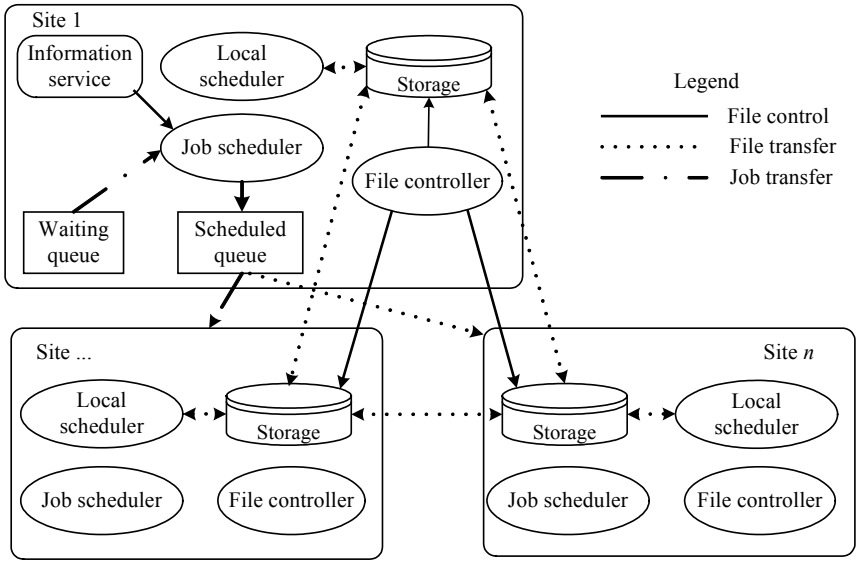

Fig.1 The resource scheduling framework

(2) Executing the job on the remote resources and obtaining its status.

(3) Transferring back output files, cleaning up the remote executing directory and related files on the user's demand.

The first and the last steps are carried out by the file controller, which can do more validation checking on the local and remote file systems.

If the data required by a job reside at some other nodes, the file controller can transfer the data to the execution nodes in the third-party transfer mode. With the third-party transfer, a job can be submitted from Location *A*, use the data at Location *B*, and be executed at Location *C*, while the results can be transferred to Location *D*. Thereby, the data, software, storage and computing resources at different locations can be shared easily.

Within this framework, when a job needs to be executed in the computational grid, it will be added to the WQ and forwarded to the JS for allocating the resources. Given the available resources information from the IS, the JS will return the best resources using the scheduling algorithm and assign them to the job, and then the job will be added to the SQ and submitted to the selected resources for execution. When a job is submitted, the file controller will be called to be in charge of file operations. And finally the job will be moved to the FQ once finished.

HEURISTIC-BASED GREEDY SCHEDULING ALGORITHM

Problem statement

The goal of the HGSA is to improve the flexibility and convenience of the resource scheduling algorithm to meet the requirements of diverse applications. Almost all the resource attributes can affect selection of the resources. Because different applications have different resource requirements, we cannot use the same rule to order the resources for different types of applications. For example, the computation-intensive applications need fast calculation speed, while the communication-intensive applications need high-speed network. Therefore, the ordering rules should be different: the CPU frequency, CPU load and available memory are more important to the former, while the network bandwidth is more important to the latter.

From the analysis above, it is apparent that one algorithm should use as many metrics as possible to schedule the resources in order to adapt to various applications, and that the metrics should have different effects on different applications to apply the same algorithm to various applications.

The metrics involved can be any available ones, which can affect the selection of the resources. At present, the attributes used include CPU count, CPU speed, CPU free rate, memory size, free memory size, file system size, free file system size, network bandwidth, latency and the allocated process count of the current job. The metrics used by the system can be extensible. Any new metrics that are attractive to the user can be added to rank the resources, such as the cost, predicted metrics, etc.

We use a metric weight to distinguish the effect of the metric on the application. The higher the metric weight value is, the more effect it has on the application.

Furthermore, the parallel job often has many processes. The assigned process of the current job may affect the subsequent resource selection for the same job. For example, the more the processes allocated, the less the free memory is, i.e., the effect is negative. If there is data exchange among processes, the processes should tend to be allocated to the same resource to speed-up the data exchange, the effect is positive. We use the metric workload impact factor to identify the effect of the allocated process on the resources selection.

HGSA is designed to follow an adaptable scheduling policy, which is defined by the weights and the workload impact factors of the metrics. In order to make the algorithm adaptive to different applications, what the user needs to do is to custom the metric weights and the metric workload impact factors according to the characteristics of the applications, that is, to define the scheduling policy.

Heuristic-based greedy scheduling algorithm

The heuristic knowledge is the metric weights of the computing resources and the metric workload impact factors.

Assume that *ResList* is the filtered available resources list; *PROCESSCOUNT* is the process count of the job, which is defined by the user; *SelectedList*

is the selected resources list, i.e. the result. The data type of *ResList* is the list of the resource classes. There are two important properties in the resource class: *processcount* and *cpucount*. The *processcount* is the allocated process count of the job to the resource, and its initial value is zero. Therefore, if the *processcount* of a resource is bigger than zero at the end of HGSA, the resource is selected. The *cpucount* is the CPU count of the resource. If the *PROCESSCOUNT* value is more than 1, the job is parallel. *ResList* and *PROCESSCOUNT* are the inputs, while *SelectedList* is the output of the algorithm. The HGSA is described in Fig.2.

```
SelectedList=NULL;
For (i=0; i<PROCESSCOUNT; ++i) {
   For each resource Res in ResList {
      If processcount<cpucount
         Calculate Rank(Res);
   }
   Select the resource with the max Rank
   value and add 1 to its processcount;
}
For each resource Res in ResList {
   If its processcount>0
      SelectedList=SelectedList+Res;
}
Return SelectedList;
```

Fig.2 The heuristic-based greedy scheduling algorithm

The mechanism of ranking the filtered resources is based on the following formula:

$$Rank=\sum_{i=0}^{k-1}[(a_i+nf_i)\omega_i],\quad \sum_{i=0}^{k-1}\omega_i=1,\ 0\le\omega_i\le 1,$$

where k is the count of metrics, n is the allocated process count of the current job, a_i is the metric value, f_i is the workload impact factor and ω_i is the metric weight of the ith metric.

In this formula, a_i is normalized. If the effect of metric i on selecting resources is negative, a_i should be $-a_i$, such as the metric of network latency.

According to Fig.2, the HGSA integrates the resource selection and workload allocation in the same process. Whether a resource is selected is determined by its property of *processcount*, which is the workload of the resource for the current job.

In practice, in order to get the best resources for an application, the user must consider the application's characteristic and customize the heuristic knowledge accordingly.

The weight of CPU speed should be increased for the computation-intensive applications due to their high requirement for computing power, while the weight of network bandwidth should be decreased. The reversed adjustment could be performed for the communication-intensive applications.

If the processes are independent within the same job, the metric of the allocated process count has no positive effect on the resources selecting, hence its workload impact factor can be zero. If the processes need to exchange data, then the metric of the allocated process count has positive effect on the resources selecting and its workload impact factor can be greater than zero.

In order to customize the metric weight and the metric workload impact factor, we use the XML file to store them. The user can edit the file to adapt to various applications.

EXPERIMENTS

We have crafted a simulation with the SimGrid (Casanova, 2001; Legrand *et al*., 2003) simulator to evaluate the HGSA. The major reasons for relying on SimGrid are that it can simulate the multi-scale computational grid and that the simulation environment can be similar to different experiments, which is important for evaluating the scheduling algorithms.

In the experiments, we use a platform described in the SimGrid software package, which consists of 90 resources, where the power ranges from 171.667 to 22.151, the bandwidth ranges from 255.228625 to 0.117125 and the latency ranges from 0.295890617 to 0.000006406. All the units omitted here adopt the ones utilized in the SimGrid.

In the experiments, we use the HSSA (shown in Fig.3), which randomly selects resources from a pool of resources and was adopted earlier in our project, to compare with the HGSA. For the HSSA we perform ten experiments in every type of experiments and use the average value as the final value.

Since the platform described in the present SimGrid is well suited for simulating the CPU power, bandwidth and latency, the experiments only use these three metrics.

//*cpucount* is the processor count of a resource.
//*P* is the resources pool, i.e. the resources set meeting the requirements.
//*S* is the selected resources set, i.e. the result.
//*n* is the process count of a job.
(1) *S*=NULL; *P*=NULL.
(2) Build *P* such that *cpucount*>=*n*, for all resources in *P*.
If *P* is not empty then select P_i from *P* at random, add *n* to the *processcount* of P_i, $S=S+P_i$, goto (5);
If *P* is empty, *i*=0.
(3) Randomly get *m*, *m*<*n*−*i*, Build *P* such that *cpucount* >=*m*, for all resources in *P*.
If *P* is not empty then select P_i from *P* at random, set *k* to be the *cpucount* of P_i, add *k* to the *processcount* of P_i, $S=S+P_i$, $n=n-k$, if *n*=0 then goto (4), if *n*>0 then goto (1);
If P is not empty then $i=i+1$, goto (2).
(4) Goto (1).
(5) Return *S*.

Fig.3 The heuristic-based stochastic scheduling algorithm

In order to present the HGSA is adaptive, we devise three sets of experiments, which simulate different types of applications. The first set of experiments only deals with computation, which simulated the computation-intensive applications. The computation size ranges from 500 to 5000 with the step size being 500. The second set of experiments only deals with communication, which simulated the communication-intensive applications. The communication size ranges from 100 to 1000 with the step size being 100. There are both computation and communication in the third set of experiments, in which the computation size and communication size follow the ones in the first two sets.

In the first set of experiments, because the applications are computation-intensive, the CPU speed weight is 1, the other metric weights are 0 and the metric workload impact factors are all set to zero. The experimental result is illustrated in Fig.4a. Because the selected resources are the fastest with the HGSA, the run time is the shortest.

In the second set of experiments, because the applications only deals with communication, the bandwidth weight is 0.9, the latency weight is 0.1, while the workload impact factor of the process count is 1. The experimental result is illustrated in Fig.4b. It is obvious that the resources selected by HGSA have good communication performance, so the run time of the applications is relatively short.

In the third set of experiments, the CPU speed weight is 0.6, the bandwidth weight is 0.35, the latency weight is 0.05, and the workload impact factor of the process count is 0.5. The experimental result is depicted as Fig.4c. Because the HGSA can take both computation and communication into account, the applications in this set of experiments can get better performance.

Three conclusions can be made from the above experiments.

(1) The HGSA does a better job than the HSSA. The makespans of the HGSA are lower than those of the HSSA in the above figures.

(2) The performance of the HGSA is stable. The curve of makespan for the HGSA is almost linear, while the curve of makespan for the HSSA is wave-like.

(3) The HGSA is adaptive to different types of applications. The HGSA can do good jobs in these three sets of experiments.

In order to demonstrate the performance and the flexibility of the HGSA, we do further experiments on the basis of the third set of experiments. We choose the resource with the fastest CPU using the scheduling policy adopted in the work of (Chapin and Spafford, 1994) and the resource with the fastest network using the ad-hoc greedy approach presented in (Petitet *et al*., 2001) to execute the job in the third set of experiments respectively. In practice, we use the HGSA to simulate their implementation. The experiment result is illustrated in Fig.4d.

Because the job in the third set of experiments involves both computation and communication, if we are only concerned with the performance of the CPU or the network, we cannot get the best result. In the same way, the single scheduling policy cannot adapt to different types of applications. With the HGSA, the users can custom the scheduling policy for the application, so that it is adaptive to various applications.

CONCLUSTION AND FUTURE WORK

In the computational grid, selecting appropriate resources for a specific application is challenging. Resource ranking is always based on resource metrics. The rule to rank the resources varies with the appli-

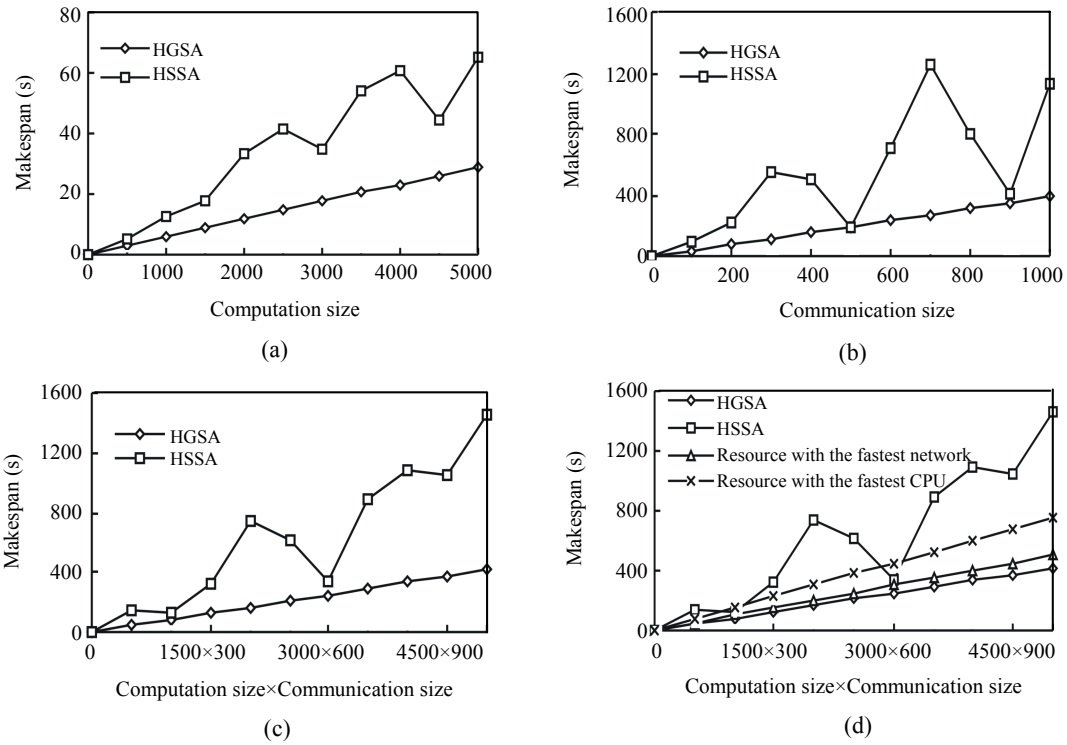

Fig.4 The experiment results. (a) Comparison between HSSA and HGSA for computation-intensive applications; (b) Comparison between HSSA and HGSA for communication-intensive applications; (c) Comparison between HSSA and HGSA for with both the computation and the communication applications; (d) Comparison for different scheduling algorithms for with both the computation and the communication applications

cation, which is also the case for the metrics involved and their functions. Therefore, resource scheduling should be application-oriented.

We have presented a distributed scheduling framework that provides a common resource selection service for different kinds of applications. As a hierarchy structure, it can consider all the available resources on the top level and select the appropriate ones according to the scheduling policy. In this framework we adopt HGSA, a heuristic greedy scheduling algorithm, to select appropriate resources for the applications. In order to be adaptive to different types of applications, the algorithm uses the metric weights of the computing resources and the metric workload impact factors to reflect the resources requirements of the applications. SimGrid was employed to simulate the grid environment to illustrate the adaptability and validity of the HGSA in the presented framework, with promising results being obtained in our project.

The contributions of this paper are as follows. We present a novel approach to model the resource scheduling problem adaptive to different kinds of applications in the computational grid environment. Using our approach, the system can assign the workload when selecting resources. Furthermore, this method is easy to use for the domain scientists.

In the future, we plan to create more practical platforms to investigate its performance and improve the algorithm in practice. Moreover, we plan to study how to automatically produce the metric weights and the metric workload impact factors according to the characteristic of the applications.

ACKNOWLEDGEMENT

We appreciate helpful discussions among the members of the Grid Computing Group at the Center for Engineering and Scientific Computation (CESC), Zhejiang University, and would like to thank Chaoyan Zhu and Wei Wang for their work in the project. We also thank Bangti Jin for his valuable discussions.

References

Aggarwal, A.K., Kent, R.D., 2005. An Adaptive Generalized Scheduler for Grid Applications. Proceedings of the 19th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'05). Guelph, Ontario, Canada, p.15-18.

Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., *et al*., 2003. Adaptive computing on the grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, **14**(4):369-382. [doi:10.1109/TPDS.2003.1195409]

Casanova, H., 2001. Simgrid: A Toolkit for the Simulation of Application Scheduling. Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01). IEEE Computer Society, p.430-437.

Casanova, H., Obertelli, G., Berman, F., Wolski, R., 2000. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. Proceedings of Supercomputing 2000. IEEE Computer Society Press, Dallas, USA, p.75-76.

Chapin, S.J., Spafford, E.H., 1994. Support for implementing scheduling algorithms using MESSIAHS. *Scientific Programming*, **3**:325-340.

Foster, I., Kesselman, C., 1998. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, San Francisco, CA, USA.

Foster, I., Kesselman, C., Tuecke, S., 2001. The anatomy of the grid: enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, **15**(3):200-222.

Gao, Y., Rong, H.Q., Huang, J.Z.X., 2005. Adaptive grid job scheduling with genetic algorithms. *Future Generation Computer Systems*, **21**(1):151-161. [doi:10.1016/j.future.2004.09.033]

Huedo, E., Montero, R.S., Llorente, I.M., 2004. Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications. Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP'04). A Coruña, Spain, p.28-33. [doi:10.1109/EMPDP.2004.1271423]

Jin, H., Shi, X., Qiang, W., Zou, D., 2005. An adaptive meta-scheduler for data-intensive applications. *International Journal of Grid and Utility Computing*, **1**(1):32-37. [doi:10.1504/IJGUC.2005.007058]

Legrand, A., Marchal, L., Casanova, H., 2003. Scheduling Distributed Applications: The SimGrid Simulation Framework. Proceedings of the 3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03). Tokyo, Japan, p.138-145.

Liu, C., Yang, L.Y., Foster, I., Angulo, D., 2002. Design and Evaluation of a Resource Selection Framework for Grid Applications. Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11). IEEE CS Press, p.63-72.

Petitet, A., Blackford, S., Dongarra, J., Ellis, B., Fagg, G., Roche, K., Vadhiyar, S., 2001. Numerical libraries and the grid. *The International Journal of High Performance Computing Applications*, **15**(4):359-374. [doi:10.1177/109434200101500403]

Ranganathan, K., Foster, I., 2002. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11). IEEE CS Press, p.352-358.

Yang, L.Y., Schopf, J.M., Foster, I., 2003. Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments. Proceedings of Supercomputing 2003. ACM Press, Phoenix, AZ, USA, p.31-46.

YarKhan, A., Dongarra, J.J., 2002. Experiments with Scheduling Using Simulated Annealing in a Grid Environment. Third International Workshop on Grid Computing. LNCS 2536, p.232-242.