Improve the Performance of Data Grids by Cost-Based Job Scheduling Strategy

N. Mansouri
Department of Computer Science, Shahid Bahonar University of Kerman, Postal Code 97175-569, Kerman, Iran
najme.mansouri@gmail.com

ABSTRACT

Grid environments have gain tremendous importance in recent years since application requirements increased drastically. The heterogeneity and geographic dispersion of grid resources and applications places some complex problems such as job scheduling. Most existing scheduling strategies in Grids only focus on one kind of Grid jobs which can be data-intensive or computation-intensive. However, only considering one kind of jobs in scheduling does not result in suitable scheduling in the viewpoint of all system, and sometimes causes wasting of resources on the other side. To address the challenge of simultaneously considering both kinds of jobs, a new Cost-Based Job Scheduling (CJS) strategy is proposed in this paper. At one hand, CJS algorithm considers both data and computational resource availability of the network, and on the other hand, considering the corresponding requirements of each job, it determines a value called W to the job. Using the W value, the importance of two aspects (being data or computation intensive) for each job is determined, and then the job is assigned to the available resources. The simulation results with OptorSim show that CJS outperforms comparing to the existing algorithms mentioned in literature as number of jobs increases.

Keywords: Data Grid, Scheduling, Access Pattern, Scheduling, Simulation.

1. INTRODUCTION

In scientific environments such as High Energy Physics (HEP), hundreds of end-users may individually or collectively submit thousands of jobs that access subsets of the petabytes of HEP data distributed over the world. Most applications especially in scientific and engineering fields tend to be data-intensive and/or computation-intensive. Due to the fact that it is impossible to manage these applications in a central server, Grid technology has been proposed as a suitable infrastructure to replace it. All jobs in such environment will compete for some resources and this is possible to distribute the load disproportionately among the Grid sites. One of the most important challenges in Grid is job scheduling problem. Indeed, determining the optimal schedule for a Grid environment which can distribute the sent jobs to the Grid resources to optimize a specify measure is a well-known NP-complete problem [1]. To overcome this difficulty, many heuristic strategies have been presented to appropriately schedule jobs among resources [2-5]. None of these types of scheduling strategies can be clearly claimed to propose optimal solution. Moreover, current scheduling strategies [6-7] are immutable to changing schedules and behave like static time-dependent Grid systems. These schedulers cannot consider the input parameters such as network features and data location at runtime. The job scheduler should take into consideration input constraints such as data location, data size, site availability, network features, computation power and various optimization criterions in making scheduling decisions. The rest of the paper is organized as follows: Section 2 introduces related work of this study. Section 3 presents the proposed job scheduling algorithms. Section 4 describes the elements of Grid simulation. We show and analyze the simulation results in section 5. Finally, section 6 concludes the paper and suggests some directions for future work.

2. RELATED WORK

Generally, job scheduling in Grid has been studied from the perspective of computational Grid [8-9]. In Data Grid, effective scheduling policy should consider both computational and data storage resources.

Foster et al. [10-11] proposed six distinct replica strategies for a multi-tier data: No Replica, Best Client, Cascading Replication, Plain Caching, Caching plus Cascading Replica and Fast Spread. They also introduced three types of localities, namely:

- Temporal locality: The files accessed recently are much possible to be requested again shortly.
- Geographical locality: The files accessed recently by a client are probably to be requested by adjacent clients, too.
- Spatial locality: The related files to recently accessed file are likely to be requested in the near future.

They evaluated these strategies with different data patterns: access pattern with no locality, data access with a small degree of temporal locality and finally data access with a small degree of temporal and geographical locality. The results of simulations indicate that different access pattern needs different replica strategies. Cascading and Fast Spread performed the best in the simulations. They have presented in another work [21] the problem of scheduling job and data movement operations in a distributed "Data Grid" environment to identify both general principles and specific strategy that can be used to improve system utilization and/or response times. They have also proposed framework with four different job scheduling algorithms, as follows:

(1) JobRadom: select a site randomly, (2) JobLeastLoaded: select a site where has the least number of jobs waiting to run, (3) JobDataPresent: select a site where has requested data, and (4) JobLocally: run jobs locally. These job scheduling strategies are combined with three various replication algorithms: (1) DataDoNothing: there is no replication and data may be fetched from a remote site for a particular job, (2) DataRandom: when popularity of the file exceeds a threshold, a replica is created at a random site, (3) DataLeastLoad: when the threshold for a file exceeds, a replica is placed at the least loaded site. They can enhance performance by scheduling jobs where data is located and using a replication policy that periodically creates new replicas of popular datasets at each site. The results also show that while it is important to consider the impact of replication on the scheduling strategy, it is not always necessary to couple data movement and computation scheduling.

Chang et al. [12] developed the Hierarchical Cluster Scheduling algorithm (HCS) and the Hierarchical Replication Strategy (HRS) to enhance the data access efficiencies in a Grid. HCS considers the locations of required data, the access cost and the job queue length of a computing node. It also takes into account hierarchical cluster Grid structure and all of data replicas owned by a cluster. The HRS replication algorithm uses the concept of “network locality” as a Bandwidth Hierarchy based Replication (BHR) strategy. HCS scheduling along with HRS replica strategy improves data access time and the amount of inter-cluster communications in comparison to others scheduling algorithms and replication strategies.

A replication algorithm for a 3-level hierarchy structure and a scheduling algorithm are proposed. Horri et al. [13] considered a hierarchical network structure that has three levels. In their proposed replication method among the candidate replicas they select the one that has the highest bandwidth to the requested file. Similarly, it uses the same technique for file deletion. This leads to a better performance comparing with LRU (Least Recently Used) method. For efficient scheduling, 3-level scheduling (3LS) algorithm selects the best region, LAN and site respectively. Best region (LAN, site) is a region (LAN, site) with most of the requested files. This will significantly reduce total transfer time, and consequently the network traffic.

Mansouri et al. [14] proposed a new job scheduling algorithm, called Combine Scheduling Strategy (CSS). CSS first selects the appropriate region, next selects the appropriate LAN in that region (i.e. available maximum requested files) and finally selects the appropriate site in that LAN by considering number of jobs waiting in the queue, location of required data and the computing capacity of sites. Simulation results show that CSS takes less job execution time than other strategies especially when number of jobs or size of the files or both increases.

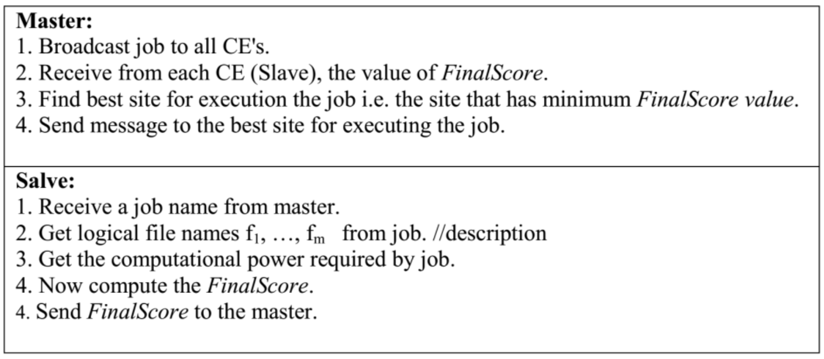
In [15] the problem of co-scheduling job dispatching and data replication in large distributed systems in an integrated manner is presented. They used a massively-parallel computation model that contains a collection of heterogeneous independent jobs with no inter job communication. The proposed model has three variables within a job scheduling system: the job order in the scheduler queue, the assignment of jobs to the nodes, and the assignment of data replicas to data stores. Finding the optimal tuple requires an exhaustive search and it is costly because the solution space is very big. The results show that deploying a genetic search algorithm has the potential to achieve better performance than traditional allocation methods.

Kumar et al. [16] showed why network characteristics, data locations of input files, and disk read speed of data sources must be taken into account when scheduling data intensive jobs, not only to minimize file staging (data transfer) time over network, but also to reduce turnaround and waiting time of jobs in Grid environment. They presented Network and Data Location Aware Scheduling (NDAS) algorithm. The presented algorithm is evaluated by improving the existing GridWay MetaScheduler with the new scheduling algorithm. The excremental results regarding the influence of the network characteristics, data locations, disk latency of data source, and jobs types variability are presented, showing that the enhanced GridWay can perform better job scheduling resulting to lower data transfer and turnaround time.

Although some previous works have done that, such as providing shorter mean job time and higher network usage, they did not consider both types of jobs simultaneously. Therefore, CJS algorithm is proposed to improve this weakness.

3. NETWORK AND DATA LOCATION AWARE SCHEDULING (NDAS) ALGORITHM

To select a best site, a parallel strategy is proposed as shown in Fig 1.



```
Master:
1. Broadcast job to all CE's.
2. Receive from each CE (Slave), the value of FinalScore.
3. Find best site for execution the job i.e. the site that has minimum FinalScore value.
4. Send message to the best site for executing the job.

Salve:
1. Receive a job name from master.
2. Get logical file names f1, …, fm   from job. //description
3. Get the computational power required by job.
4. Now compute the FinalScore.
4. Send FinalScore to the master.
```

FIGURE 1. A parallel execution flow of master and slave.

3.1 TRANSFER TIME

Let B_{ji} is the bandwidth from site S_j to the site that f_i resides. *PropagationDelay*$_{ij}$ is propagation delay / network latency (in seconds) from site S_j to site S_i. Then transfer time for f_i (TransferTime$_{\mathrm{fi}}$) is obtained by

$$TransferTime_{fi} = PropagationDelay_{ij} + \left(|fi| \ * \ 8 \right) / B_{ji} \tag{1}$$

Let $J_x = \{f_1, f_2, .., f_m\}$ be the m required files for job x. Now estimated file staging (data transfer) time of job x when scheduled on site S_j $(JobTime_{x,j})$ is given:

$$\mathrm{JobTime}_{x,j} = \sum_{i=1}^{m} \mathrm{Min}(\mathrm{TransferTime}_i) \tag{2}$$

Replica selection is crucial to data intensive scheduling; it depends on the network characteristics and an optimized replica selection leads to an optimized data intensive scheduling. These considerations not only improved the execution times of the jobs but also reduced the queue times of the jobs. So, if several sites have the replica of f_i, it selects one that has maximum Score.

$$Score = P^{BW} \times w_1 + P^{CPU} \times w_2 + P^{IO} \times w_3 \tag{3}$$

Where P^{BW} represents the percentage of bandwidth available from the selected site to the site that requested file resides, P^{CPU} is the percentage CPU idle states of site that requested file resides, and P^{IO} is the percentage of memory free space of site that requested file resides.

$$w_1 + w_2 + w_3 = 1 \tag{4}$$

These weights can be set by the administrator of the Data Grid organization. According to different attributes of storage systems in data Grid node.

Let k is the number of jobs waiting in queue of site S_j. The value of $TotalTime_j$ for site S_j is calculated by

$$\text{TotalTime}_j = \sum_{x=1}^{k} \text{JobTime}_{x,j} \tag{5}$$

3.2 COMPUTATIONAL POWER

The processing power provided by resources (required for jobs) is described in the form of MIPS (MI). Therefore, the total time required for the job J_x to be completed in the resource S_j can be calculated by Eq. (6).

$$ComputingScore = \frac{CP_x}{CP_j} \tag{6}$$

Where CP_j is the computational power provided by the computational resource C_j and CP_x is the computational power required by the job J_x. The ComputingScore is used as a score for fitness of the resource C_j for the job J_x. The available information about each job send to the environment is stored in two areas. The first one contains information about needed data files, so we can obtain the total size of data files, and the second one gives information about the total computational power needed by the job in terms of MI. The main goal at this stage is to calculate the proportion of being data-intensive to being computation-intensive, while considering the availability of resources in each area. Hence, the strategy needs to jointly consider both required and provided resources, and then estimate a value for scheduler to show how much the submitted job is generally data/computation intensive in the context of available grid environment. To achieve this, the strategy first determines the expected value of the provided computational power using Eq. (7).

$$ComputationPower = \frac{\sum_{i=1}^{N} Cp_i}{N} \tag{7}$$

Where, N is the number of sites. To find the corresponding value for data-intensive aspect of the submitted job, the strategy needs to apply an equivalent mean operation on network links. Eq. (5) obtains this value by averaging on time needed to collect a specific set of data files for each site.

$$TotalTransferTime = \frac{\sum_{i=1}^{N} TotalTime_i}{N} \tag{8}$$

3.3 FINAL COST

Finally, the factor W is determined by using Eq. (9) and Eq. (10) for a given job i.

$$CC = \frac{CP_i}{ComputationPower} \tag{9}$$

$$TT = \frac{TotalTime_i}{TotalTransferTime}$$

$$W = \frac{CC}{CC + TT} \tag{10}$$

When the CJS strategy is executed for a submitted job, both *TotalTime* and *ComputingScore* are determined for each site. Combining these two scores by affecting the factor *W* gives the *FinalCost* for all sites (Equation 11).

$$FinalCost(J, S) = (1 - w) \times TotalTime + w \times ComputingScore \tag{11}$$

The CJS strategy chooses the site with minimum *FinalCost* and assigns the job to it.

4. ELEMENTS OF GRID SIMULATION

We have implemented the proposed strategy using OptorSim, a simulator for Data Grids. OptorSim was presented by the European Data Grid (EDG) project [17]. It provides users with the Data Grids simulated architecture and programming interfaces to analysis and validate their strategies. In order to obtain a realistic simulated environment, there are a number of components which are included in OptorSim. These include Computing Elements (CEs), Storage Elements (SEs), Resource Broker (RB), Replica Manager (RM), and Replica Optimiser (RO). Each site consists of zero or more CEs and zero or more SEs.

5. EXPERIMENTS

In this section, network configuration and the simulation results are described.

5.1 CONFIGURATION

The study of our scheduling algorithm is carried out using a model of the EU Data Grid Testbed [17] sites and their associated network geometry as shown in Fig. 2. Initially all jobs are placed on CERN (European Organization for Nuclear Research) storage element. CERN contains original copy of some data sample files that cannot be removed. Since all files are available in Site 0, so any job sent to this site does not require any file transfer. Therefore in our simulation we only consider all CE sites except site 0. Each file is set to be 1 GB. To record file transfer time and path, we changed OptorSim code. A job will typically request a set of logical filename(s) for data access. The order in which the files are requested is specified by the access pattern. We considered four different access patterns: sequential (files are accessed in the order stated in the job configuration file), random (files are accessed using a flat random distribution),Gaussian random walk (files are accessed using a Gaussian distribution), and Random Zipf access (given by $P_i = K/ i^S$, where P_i is the frequency of the *i*th ranked item, *K* is the popularity of the most frequently accessed data item and *S* determines the shape of the distribution).

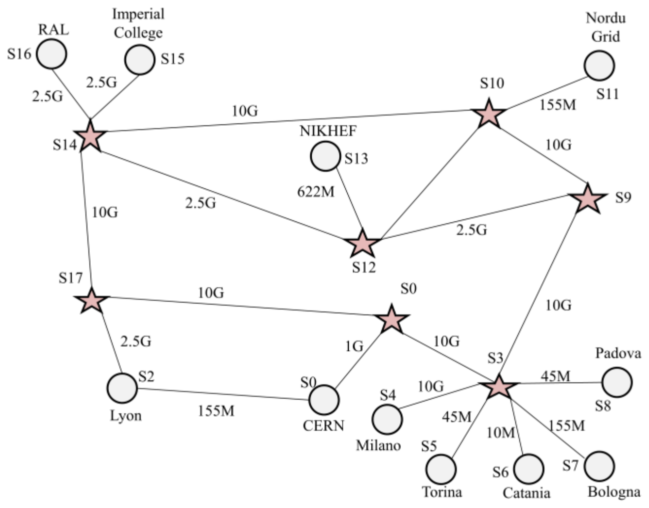


FIGURE 2. The gird topology of EDG.

5.2 SIMULATION RESULTS AND DISCUSSION

Eight scheduling strategies have been considered, as follows:

- The Random scheduler that schedules a job randomly.
- The Shortest Queue scheduler that selects computing element that has the least number of jobs waiting in the queue.
- The Access Cost scheduler that assigns the job to computing element where the file has the lowest access cost (cost to get all unavailable requested data files needed for executing job).
- The Queue Access Cost scheduler that selects computing element with the smallest sum of the access cost for the job and the access costs for all of the jobs in the queue.
- Hierarchical Cluster Scheduling (HCS) takes into account hierarchical cluster Grid structure and all of data replicas owned by a cluster. It schedules jobs to certain specific sites and specific cluster according to inter-cluster communication costs.
- 3-level Scheduling (3LS) determines most appropriate region, LAN and site respectively. An appropriate region (LAN, site) is a region that holds most of the requested files (from size point of view). i.e. most of the requested files are available in that region.
- Network and Data location Aware Scheduling (NDAS) takes into account network characteristics, data locations of input files, and disk read speed of data sources in scheduling decision.
- The Combine Scheduling Strategy (CSS) considers the number of jobs waiting in queue, the location of required data for the job and the computing capacity of sites.

Figure 3 depicts the Mean Job Time for different job scheduling algorithms with various access patterns. The mean job execution time is defined as the total time to run all the jobs divided by the number of jobs finished. The total time includes the time that elapses from when a job enters the queue in a site to await execution until the time when the job completes its processing and leaves the site. In Random scheduling the mean job execution time obviously increases because it doesn't consider any factors.

In Shortest Job Queue Scheduling each CE receives approximately the same number of jobs. If CE's have low network bandwidth, then file transfer time will be high and overall job execution time will increase. Access Cost Scheduling selects a CE based on its access cost. CE's with lower access cost may receive large number of jobs to execute. So, overall performance is decreased. The Queue Access Cost considers not only shortest job queue but also access cost. Therefore, the Queue Access Cost decreases total job execution time. The mean job time is about 8% faster using HCS than using Queue Access Cost because HCS uses a hierarchical tree to schedule a job and minimize the overhead of searching for the suitable site. The 3LS first selects the appropriate region (i.e. available maximum requested files), next selects the appropriate LAN in that region and finally selects the appropriate site in that LAN, therefore job execution time decreases since it has minimum data transfer time. The mean job time is about 12% faster using CSS than using HCS because it schedules jobs close to the data whilst ensuring sites with high network connectivity are not overloaded and sites with poor connectivity are not left idle. It also takes into account hierarchical Grid structure and considers computational capability. The mean job time of CJS is lower about 11% compared to the CSS algorithm. The reason is that it takes into account data, processing power and network characteristics when making scheduling decisions across different sites.

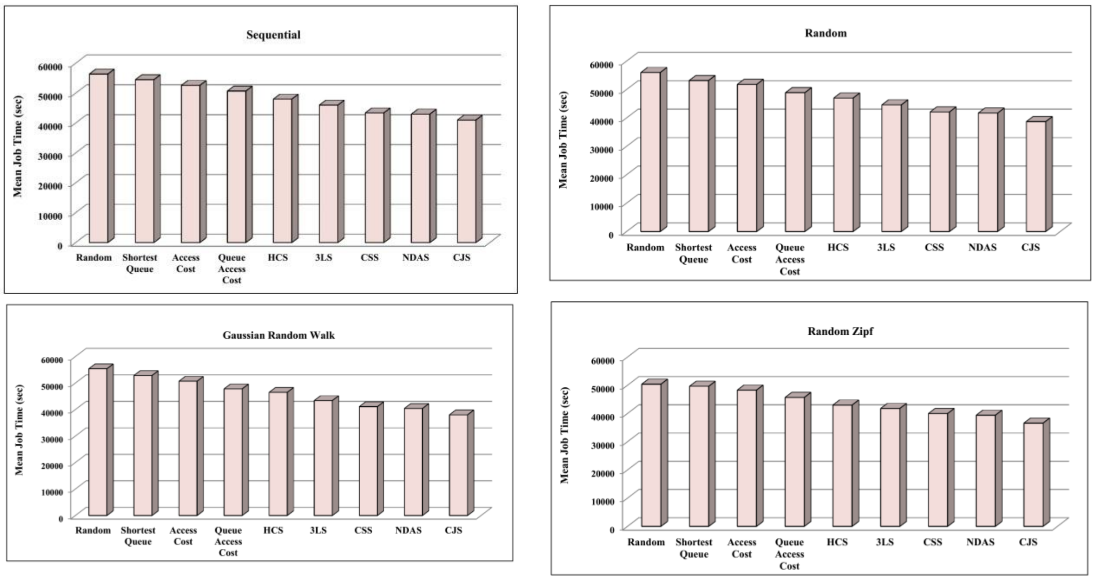


FIGURE 3. Mean job Time for different access patterns.

Figure 4 shows the queue time for nine scheduling strategies with different number of jobs. We changed the number of jobs for two important reasons: to monitor how the queue size increases over time and in which proportion the scheduler submits the jobs (that is whether the jobs are sent to some particular site or to a number of CPUs at various locations depending on the queue size and the computing capability).

It presents that queue time is almost proportional to execution time because if the job is executing and taking more time on the processor, the waiting time of the new job will also increase correspondingly since it will waste more time in the queue. Although the execution time does not comprise queue times, a higher number of jobs executing at a site can influence the queue time. Moreover, increasing the number of jobs in the queue can affect the overall job completion times (i.e. the scheduling time, queuing time and execution time) of the new jobs. The queue time of the schedulers is very important in the Grid environment and it takes a large ratio of the job's overall time. Sometimes this is greater than the execution time if the resources are rare compared to the job frequency. In experimental setup of this work, we took only a single job queue and we considered that all jobs have the same priority.

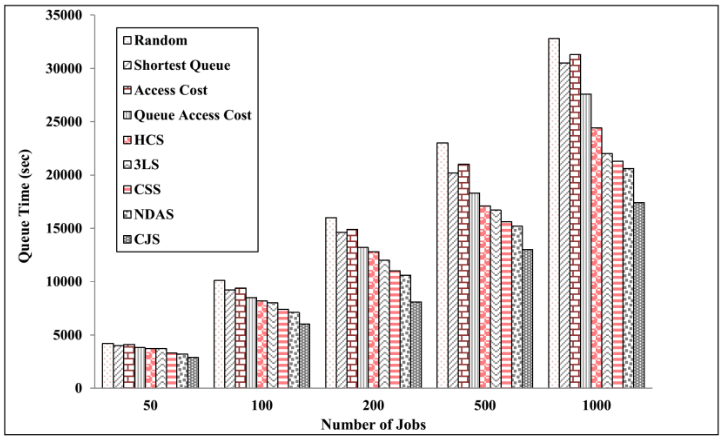


FIGURE 4. Queue time versus number of jobs.

Multi-queue and multi-priority job scenarios will be discussed later in future work. Figure 4 indicates that the queue grows with an increasing number of jobs and that the number of jobs waiting for the allocation of the processors for running also increases. From the figure it is clear that the CJS scheduling strategy remarkably decreases the queue time of the jobs. The main reason is only those sites were selected for job placement which had fewest jobs in the queue and which were likely to quickly run the jobs once scheduled on that site, were selected for job placement.

Figure 5 indicates execution times for various scheduling strategies. We see from the results obtained in Fig. 4 and 5 that both queue and execution times follow very similar trends. This is mainly due to the fact that CJS preferentially chose those sites for job execution which could execute jobs fast.

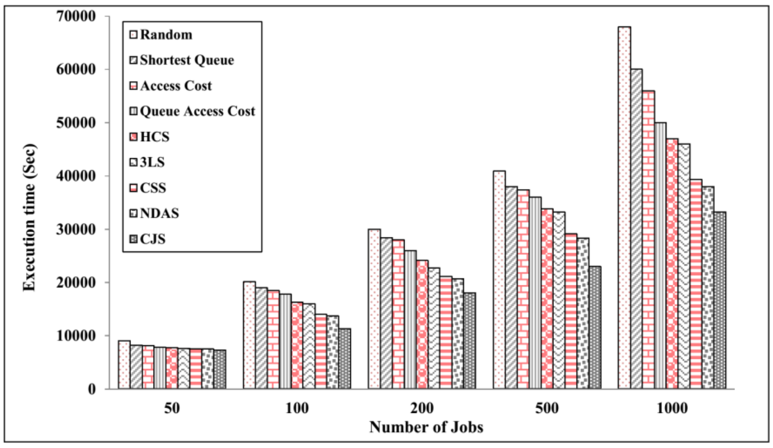


FIGURE 5. Execution time versus number of jobs.

The computing resource usage is shown in Fig. 6. It is the percentage of time that CEs are in active state. The CJS has good computing resource usage because it completes all jobs first, so the CPUs are not idle most of the time. It can make intelligent decisions by considering the changing state of the network, and the pool of processing cycles.

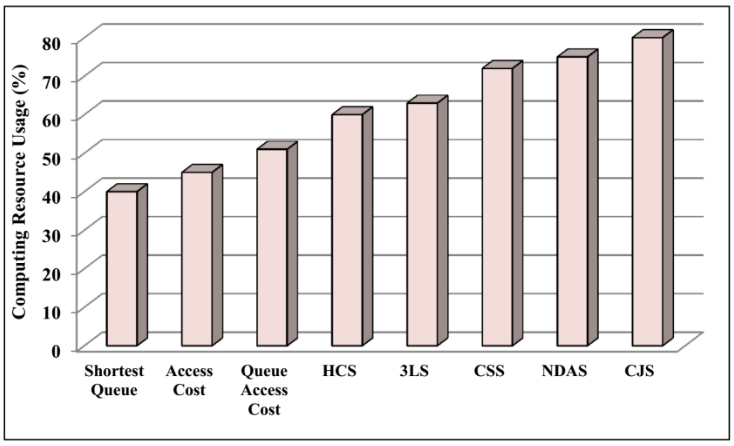


FIGURE 6. Computing resource usage for various job scheduling.

7. Conclusion

Yet effective scheduling in data grid environments is challenging, due to a need to address a variety of metrics and constraints (e.g., resource utilization, response time,) while dealing with multiple, potentially independent sources of jobs and a large number of storage, compute, and network resources. Considering various requirements of jobs during scheduling decision within Grid environments is the main concern of this paper. The scheduler can make “intelligent” decisions by taking into account the changing state of the network, the locality and the size of the data and the computational power. To achieve a more appropriate scheduling in Grids, an algorithm named CJS is proposed in this paper to discuss the problem of simultaneously considering data-intensive and computation-intensive dimensions of the jobs.

The CJS strategy takes network characteristics as a primary class criterion in the scheduling decision, along with computations and data. It was also deduced that a combination of data transfer cost, network cost and computation cost can considerably optimize the Grid scheduling and execution process which was the key message of the CJS scheduling approach. A grid simulator (i.e. OptorSim) was utilized to evaluate the CJS algorithm. The simulation results showed that the new algorithm enhanced the performance of the grid environment and thus, decreased the job’s average total time. From a simulation perspective, it will be interesting to evaluate the results in more complex networks. Another interesting issue, is modeling a real grid scenario, with the existing resources and real job traces.

REFERENCES

[1] D. Fernandez-Baca, “Allocating modules to processors in a distributed system,” *IEEE Transactions on Software Engineering*, vol. 15, pp. 427-1436, 1989.

[2] S. Kardani-Moghadam, F. Khodadadi, R. Entezari-Maleki and A. Movaghar, “A hybrid genetic algorithm and variable neighborhood search for task scheduling problem in grid environment,” *Procedia Engineering*, vol. 29, pp. 3808-3814, 2012.

[3] R. Entezari-Maleki and A. Movaghar, “A genetic-based scheduling algorithm to minimize the makespan of the grid applications”, *in: Grid and Distributed*

Computing Conference , *Communications in Computer and Information Science (CCIS)*, pp. 22-31, 2010.
[4] Z. Mousavinasab, R. Entezari-Maleki and A. Movaghar, "A bee colony task scheduling algorithm in computational grids*," in: Iternational Conference on Digital Information Processing and Communications (ICDIPC)*, pp. 200-211, 2011.
[5] B. Radha, V. Sumathy, "Enhancement of grid scheduling using dynamic error detection and fault tolerance," *International Journal of Computer Applications*, vol. 31(7), 2011.
[6] J. Nabrzyski, J.M. Schopf, and J. Weglarz, "Grid Resource Management," *Kluwer Publishing*, 2003.
[7] L.R. Anikode and B. Tang, "Integrating scheduling and replication in data grids with performance guarantee," *in: Global Telecommunications Conference*, pp. 1-6, 2011.
[8] R.S. Chang, C.Y. Lin and C.F. Lin, "An adaptive scoring job scheduling algorithm for grid computing," *Inform Sciences*, vol. 207, pp. 79-89, 2012.
[9] A. Chaudhuri, D. Jana and B.B. Bhaumik, "Optimal model for scheduling of computational grid entities," *in: India Conference (INDICON)*, pp. 1-6, 2011.
[10] I. Foster and K. Ranganathan, "Design and evaluation of dynamic replication strategies for high performance data grids," *in: Proceedings of International Conference on Computing in High Energy and Nuclear Physics*, 2001.
[11] I. Foster and K. Ranganathan, "Identifying dynamic replication strategies for high performance data grids," *in: Proceedings of 3rd IEEE/ACM International Workshop on Grid Computing*, pp. 75–86, 2002.
[12] R. Chang, J. Chang, and S. Lin, "Job scheduling and data replication on data grids," *Future Gener Comp Sy*, vol. 23, pp. 846-860, 2007.
[13] A. Horri, R. Sepahvand and G.H Dastghaibyfard, "A hierarchical scheduling and replication strategy," *International Journal of Computer Science and Network Security*, vol. 8, 2008.
[14] N. Mansouri, G.H. Dastghaibyfard and E. Mansouri, "Combination of data replication and scheduling algorithm for improving data availability in Data Grids," *J Netw Comput Appl*, vol. 36, pp. 711-722, 2013.
[15] S. Vazhkudai, "Enabling the co-allocation of grid data transfers*," in: Proceedings of the Fourth International Workshop on Grid Computing*, pp. 44-51, 2003.
[16] S. Kumar and N. Kumar, "Network and data location aware job scheduling in grid: improvement to GridWay meta scheduler," *International Journal of Grid and Distributed Computing*, vol. 5(1), 2012.
[17] D.G. Cameron, A.P. Millar, C.C. Nicholson, R. Carvajal-Schiaffino, F. Zini and K. Stockinger, "Optorsim: a simulation tool for scheduling and replica optimization in data grids," *in: International conference for computing in high energy and nuclear physics (CHEP'04),* 2004.