CO-ALLOCATION IN GRID COMPUTING USING RESOURCES OFFERS AND ADVANCE RESERVATION PLANNING

SID AHMED MAKHLOUF & BELABBAS YAGOUBI

Dept. of Computer Science, University of Oran, Algeria
sidahmed.makhlouf@gmail.com, byagoubi@gmail.com

ABSTRACT

Computational grids have the potential for solving large-scale scientific problems using heterogeneous and geographically distributed resources. However, a number of major technical hurdles must overcome before this potential can be realized. One problem that is critical to effective utilization of computational grids and gives a certain Quality of Service (QoS) for grid users is the efficient co-allocation of jobs. The advance reservation technique has been widely applied in many grid systems to provide QoS, however, it will result in low resource utilization rate and high rejection rate when the reservation rate is high. This work addresses those problems by describing and evaluating a grid resources co-allocation algorithm using resources providers offers and planning the advance reservations. In our algorithm, a Metascheduler performs job scheduling based on resources offers and use advance reservation planning mechanism to reserves the best offers. Offers act as a mechanism in which resource providers expose their interest in executing an entire job or only part of it. The Metascheduler selects computational resources based on best offers provided by the resources; Meta-schedulers can distribute a job among various clusters that are usually heterogeneous in order to speed up the job execution.

The main aims of our algorithm is to minimize the total time to execute all jobs (Makespen), minimize the waiting time in the global queue, maximize the resources utilization rate and balance the load among the resources. The proposed algorithm has been compared with other scheduling schemes such as First Come First Served (FCFS), easy backfilling (EBF), Fit Processor First Served (FPFS) and a simple co-allocation algorithm without offers support (SCOAL). The proposed algorithm has been verified through an extension of GridSim simulation toolkit and the simulation results confirm that the proposed algorithm allow us to achieve our goals by minimizing the Makespan and the waiting time, maximizing the resources utilization rate and load the balance among the resources.

KEY WORDS*:* Grid Computing, Grid Scheduling, Resources Allocation, Resources Co-allocation

1 INTRODUCTION AND RELATED WORK

Grid Computing is concerned with "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations" [11]. The coordination between multiple administrative domain results heterogeneity in Grid Environment. The resources in Grid Computing include supercomputers, workstations, databases, storages, networks and so on. Resources owned by various administrative organizations are shared under locally defined policies that specify what is shared, who is allowed to access what, and under what conditions [12]. To achieve the promising potentials of computational Grids, an effective and efficient scheduling system is fundamentally important. Scheduling (or resources allocation) in Grid environments is significantly complicated by the heterogeneous and dynamics nature of Grids. Compared to traditional scheduling systems such as cluster computing, Grid scheduling systems have to take into account diverse characteristics of both various Grid applications (jobs) and various Grid resources (resources providers). The different performance goals also place great impacts on the design of scheduling systems. Sometimes, the needs of a single job may exceed the capacity available in each of the subsystems (i.e. clusters) making up a grid, and so co-allocation (i.e. the simultaneous access to resources of possibly multiple types in multiple locations managed by different resource managers or locals scheduler) may be required.

Various co-allocation frameworks have been developed in various grid systems [8,22]. These frameworks mainly focus on co-allocation service architecture and programming interfaces for grid applications.

As complementary to co-allocation frameworks, many co-allocation policies and models were proposed. LEINBERGER et al [16] proposed two backfilling-based heuristics for K-resource co-allocation. Their studies showed that load-balancing policy outperforms classical policies over 50% in terms of mean response time.

MOHAMED and EPEMA [17] proposed a close-to-files

policy, which tried to place jobs on clusters closing to the input files so as to reduce communication overhead. HE et al [14] proposed two policies (ORT and OMT) to address co-allocation. ORT aimed to optimize mean response time for non-real-time jobs, while OMR was designed to achieve the optimal mean deadline miss rate for soft real-time job streams. To evaluate the performance of various co-allocation policies, BUCUR and EPEMA [3,4] conducted extensive experiments in large-scale grid system. Based on their experimental results on grid test-bed DAS-2 [2], they made an important conclusion that workload-aware policies were effective to reduce the mean response time and obtain better load-balance. Unfortunately, most of the above policies aimed to improve the system performance metrics, but few of them took the users' QoS constraints into account.

A grid economy [6] has been introduced into grid systems, such as Spawn [21], Popcon [18], and Nimrod-G [1]. For instance, Nimrod-G provided three adaptive policies for deadline and budget constrained resources allocation [5]: cost optimization, time optimization, and conservative time optimization. Although economic model has been proven to be an effective method for resource allocation in distributed environments, it has two shortcomings that cannot be ignored: economic models bring about extra communicational and computational overhead to applications [6]; and in presence of high-end applications that require co-allocating multiple resources across sites, the price negotiation process is often low-efficient [5].

Advance reservation as an effective technique to support QoS has been incorporated into many grid systems. It allows applications to gain concurrent access to adequate resources, and guarantees the availability of resources at the required time [10], however, advance reservation have many negative effects on resource sharing and jobs scheduling in the grid systems. For instance, studies in [13,20] show that the fixed capability reservation results in a low resource utilization rate, and excessive reservation often leads to a high rejection rate. These negative effects influence the grid economy [6], where resource providers wish to increase the utilization rate of their resources to obtain maximal profits.

This work focused on resources co-allocation using resource providers' offers and advance reservation planning in grid computing. A new co-allocation model had been introduced where a Metascheduler receive a job from grid user, get a different offers from resources providers to execute the job, then dispatches the job to the selected resources based on the best provided offers to improve resource and user benefit. The objectives of introducing offers with advance reservation planning to the co-allocation system are as follows: (1) improve users benefit by minimizing theirs job's Makespan and waiting time; (2) improve resources benefit by maximizing theirs utilization rate and load the balance among all the resources providers. Therefore, in this model there are three types of participants: resource providers, Metascheduler and grid users. We assume that the co-allocation algorithm is non-preemptive, and all the jobs are independent.

The remaining part of this paper is organized as follows. Our Grid Co-allocation Architecture, Model and Motivations are presented in Section 2. In Section 3, the algorithmic description of our co-allocation policy is presented. An experimental setup along with the comparative results is explained in Section 4. Conclusion and future research direction is proposed in Section 5

2 SYSTEM ARCHITECTURE, MODEL AND MOTIVATIONS

2.1 System Architecture

The system architecture is described in figure 1. The main components used in this architecture are grid users, Metascheduler and resources providers. Each resource provider may differ from the rest of the others providers with respect to number of processors, speed of processing, local scheduling, etc. A Metascheduler receives job from grid users, selects feasible resources for those job according to the bests proposed offers generated from resources providers and finally submit the job to the selected resources.

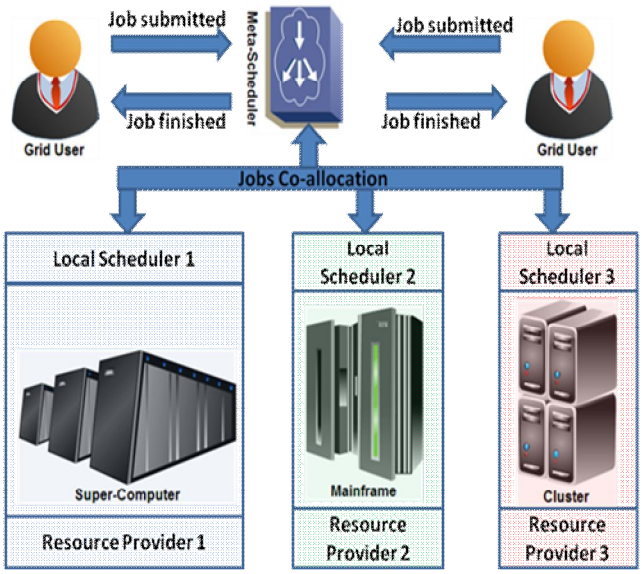

Figure 1: Co-allocation system architecture

2.2 System Model

Figure 2 presents a UML sequence diagram depicting our grid co-allocation process. The interactions are between the Grid users, the Metaschedule, and local schedulers of the resources providers. A user sends a job request to the Metascheduler for computation. Then the Metascheduler validates the request and interacts with local schedulers for requesting offers. Then it tacks the best resource offers through the procedure that is described in Algorithm 2. Finally, the submitted Job is sent to the selected resources that propose the best offers. When the job is finished, the resources send back a completion event to Metascheduler.

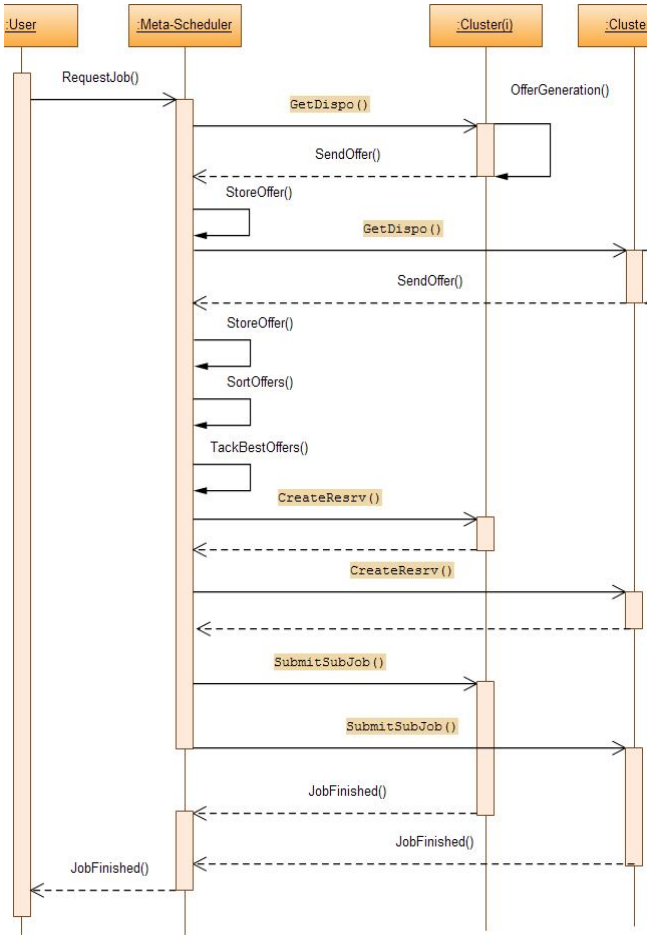

Figure 2: Sequence Diagram of co-allocation Process

2.3 Motivations

Conventionally, advance reservation is defined as a process of requesting resources for use at a specific time in the future [19]. The two key attributes of a reservation request are starting time and reservation duration time. As the availability and performance of resources are unpredictable in large-scale grid systems, precisely estimating these two parameters is difficult if not impossible. Consequently, applications tend to overestimate these two parameters (especially the reservation duration time) to ensure their successful execution [13]. This behavior results in a high rejection rate and a low resource utilization rate.

Motivated by these facts, the authors propose a co-allocation policy with offers advance reservation planning (COARP). The idea is to ask the resource providers for their free time slots in the future and reserve the best slots that cans meet the user's requirements, see figure 3.

The objective of COARP is to minimize the user's jobs response time, minimize the waiting time in the global queue, increase the resource utilization rate and load the balance among all the resource providers.

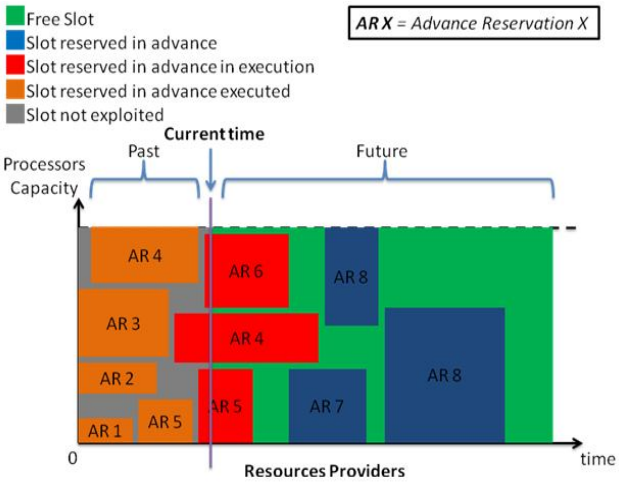

Figure 3: Resources provider's slots representation in the time

3 CO-ALLOCATION ALGORITHM

3.1 Resources Offer Generation

Offer O_c consists of a list of slots. A slot $S_{(i,c,D)}^{(s,n)}$ is windows for a resource provider c that represent it's start time availability s. Etch slot i consist of a number available processors n at time s and duration time D to execute a job. We can represent an offer from resource provider c as $O_c = \left\{ S_{(i,c,D)}^{(s,n)} \mid i = \overline{1,|O_c|} \right\}$. Offers are to execute part or the entire job. The resource provider could follow different policies to generate offers. For instance, a resource provider could generate offers that are more profitable [9,15]; provide some slack in case of resource failures or to increase the chances of admitting more jobs in future; or do not violate the reservation of already accepted jobs. In this work, we support the third approach and will leave the other two for future work. The offer generation is described in the algorithm 1.

Input : Job Request : J_L^P
Input : Resource Provider Speed : $speed$
Input : Advance Reservation List : $ARList$
Output : Offer : O_c

/* Get resource provider identifier */;
1 $c \leftarrow ResourceProviderId$;
/* Calculate the execution duration time D of the job J_L^P */;
2 $D \leftarrow \frac{L}{Speed}$;
/* Initialize Offer O_c */;
3 $O_c \leftarrow \phi$;
/* Find all the available free slots in the ARList and put them in the Offer O_c */;
4 $i \leftarrow 1$;
5 **while** *there is an empty slot in the ARList* **do**
6 $s \leftarrow getAvailableFreeStartTimeIn\,(ARList)$;
7 $n \leftarrow getAvailableFreeProcessorsAtTime\,(s)$;
8 $S_{(i,c,D)}^{(s,n)} \leftarrow ComposeFreeSlot\,(s,n,i,c,D)$;
9 $O_c \leftarrow O_c \cup \left\{ S_{(i,c,D)}^{(s,n)} \right\}$;
10 $i \leftarrow i+1$;
11 **end**
/* Finely, return the offer O_c */;
12 **Return** O_c;

Algorithm 1: Resources Offer Generation

The resource provider uses the job information provided by the Metascheduler J_L^P that includes the length L of the job in MIPS (Millions Instructions Per Second), and number of required processors P.

In order to generate an offer O_c for a job J_L^P, the resource provider c:

- Calculate the execution duration time D of the job.
- Find all the available free slots in the advance reservation list.
- Create a list of free slots O_c for each job J_L^P. Etch slot i include it's start time s, the number of the available processors n, the duration D to execute the job and the resource provider identifier c.
- Return the offer O_c to the Metascheduler.

3.2 Offers Composition

The Metascheduler is responsible for composing the offers from the different resource providers to meet a job requirement. The offer composition determines how much work the Metascheduler should send to each resource provider. The goal of the Metascheduler is to meet users' requirement, minimize the job Makespen and waiting time, maximize the resources utilization rate and load the balance among the resources providers. The offer composition is described in the algorithm 2.

Input : Job : J_L^P
Input : Numbers of resource providers : N
Output : Status of the co-allocation operation : status

/* send a availability request to all resources providers with the function *queryFreeSlot(ResourceIdentifier)* */;
$OffersList \leftarrow \phi$
for $c = 1 ; c \leq N ; c \leftarrow c + 1$ **do**
  $O_c \leftarrow queryFreeSlot\,(c)$;
  $OffersList \leftarrow OffersList \cup O_c$;
end
/* Check if the offers list is empty */;
if $OffersList = \phi$ **then**
  /* we leave the algorithm with a failure status */;
  $status \leftarrow failure$;
  Return *status*;
end
/* We sort the OffersList into the SortedOffersList by finish time */;
$SortedOffersList \leftarrow sort(OffersList)_{FinishTime}$;
/* Create BestOffersList and put in it the first best slots in the SortedOffersList according to the job requirement P */;
$BestOffersList \leftarrow \phi$;
$NumPE \leftarrow 0$;
while $NumPE < P$ **do**
  $S_{(i,c,D)}^{(s,n)} \leftarrow \min\,(SortedOffersList)_{FinishTime}$;
  $BestOffers \leftarrow BestOffers \cup \left\{S_{(i,c,D)}^{(s,n)}\right\}$;
  $NumPE \leftarrow NumPE + n$;
  $SortedOffersList \leftarrow SortedOffersList - \left\{S_{(i,c,D)}^{(s,n)}\right\}$;
end
/* for etch slot in the BestOffersList, send an advance reservation request with function *sendAR(ReservationStartTime, NumbersOfProcessorsRequested, ReservationDuration, Job, ResourceProviderIdentifier)* */;
while $BestOffersList \neq \phi$ **do**
  $S_{(i,c,D)}^{(s,n)} \leftarrow get\,(BestOffersList)$;
  $BestOffersList \leftarrow BestOfferList - \left\{S_{(i,c,D)}^{(s,n)}\right\}$;
  $sendAR\,(s, n, D, J_L^P, c)$;
end
$status \leftarrow success$;
Return *status*

Algorithm 2: Offers Composition

After collecting the offers from all the N resource providers, the Metascheduler:

- Creates a $OffersList = \bigcup_{c=1}^{N} O_c$ with all the proposed offers, etch offer O_c from resource provider c contain a set of free slots $O_c = \left\{S_{(i,c,D)}^{(s,n)} \mid i = \overline{1, |O_c|}\right\}$. If the offers list is empty, then the job will be placed in the global waiting queue and it will be rescheduled as the resources will become available again.
- Sorts the $OffersList$ in ascending order by finish time with FinishTime = StartTime + DurationTime.
- Create a $BestOffersList \subseteq OffersList$ that contains the first best slots in the $OffersList$ according to the job requirement P.
- For etch slot $S_{(i,c,D)}^{(s,n)}$ in the $BestOffersList$ with $i = \overline{1, |BestOffersList|}$, send an advance reservation request $AR\langle s, n, D, J_L^P, c\rangle$ to the resource provider c that include slot's start time s, slot's available processors n, the job execution duration time D in the resource provider c and the job request J_L^P.

4 EXPERIMENTAL SETUP AND RESULTS ANALYSTS

4.1 Experimental Setup

We have evaluated our co-allocation policy by means of simulations to observe its effect in a long-term usage. We have used the event-driven simulator named GridSim [7], which we have extended to support jobs on multi-site environments. We have used real traces from supercomputers available at the Parallel Workloads Archive. In our simulation, various entities connected by a network and every two entities' connectivity had an exclusive bandwidth. In order to make comparisons with other results we chose FCFS, EBF, FPFS and Simple co-allocation policy (SCOAL) as the benchmarks because most related works evaluated these algorithms.

We have modeled an environment composed of seven clusters with their own local schedulers, and one Metascheduler that receives jobs that can be executed in either a single or multiple clusters, Table 1 illustrate our grid environment. We have used the trace file of the San Diego Supercomputer Center Blue Horizon with 1,152 processors, 144 nodes IBM SP, with 8 processors per node node. More details on the trace file can be found at the Parallel Workloads Archive[1]. We have simulated 270 days of this traces that is equivalent to 53349 jobs.

[1] http://www.cs.huji.ac.il/labs/parallel/workload

Table 1: Computational Grid Environment

Resource Provider Name	Processors Numbers	Resource Speed (MIPS)
Cluster 1	280	1000
Cluster 2	240	1100
Cluster 3	200	1200
Cluster 4	160	1300
Cluster 5	120	1400
Cluster 6	80	1500
Cluster 7	40	1600

We performed the experiments on a PC with two Cores Processors, 2.20GHz and 2GB RAM using Linux 64 bits. We have assessed four metrics:

- Makespan: total time to execute all the submitted jobs, $Makespen = \sum_{i=1}^{MaxJobs} (F_i - E_i)$ with E_i and F_i are respectively the execution start time and the execution finish time of the job i.
- Waiting Time: average time difference between the submit time and execution start time of all the jobs, $WaitingTime = \frac{\sum_{i=1}^{MaxJobs} (E_i - S_i)}{MaxJobs}$ with S_i and E_i are respectively the submission time and the execution start time of the job i;
- System Utilization: average resource utilization rate $\omega = \frac{\sum_{c=1}^{N} \omega_c}{N}$ with N the numbers of resources providers, ω_c is the system utilization rate of resource's provider c.
- System Load balancing: standard deviation $\sigma = \sqrt{\frac{\sum_{c=1}^{N} (\omega_c - \omega)^2}{N}}$ of the resource's providers utilization rate to measure the load variations between the clusters.

4.2 Results analysts

Makespan: From Figure 4 we can see that when the number of jobs increases, the Makespan increases and by comparing the five curves of Figure 4, we see that we have obtained a gain in Makespan by using our co-allocation algorithm compared to the other scheduling algorithms. This is due to the co-allocation strategy that allows the distribution of the job's execution time on the most available resources when there is not an adequate resource, unlike the other scheduling algorithms that try to find the best resource that satisfy the job's requirements; otherwise, the job will remain in the global queue of the Metascheduler, which will delay the job's execution start time.

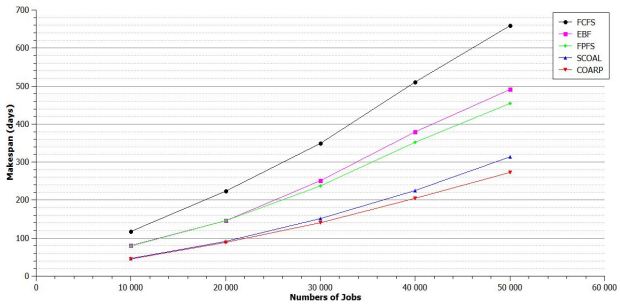

Figure 4: Total Makespan to execute all the jobs

Waiting time: From Figure 5 we can see that when the number of jobs increases, the average waiting time decreases and by comparing the five curves of Figure 5, we see that our co-allocation algorithm allowed us to reduce the average waiting time of the jobs compared to the other scheduling algorithms. This is due to the advance reservation that ensures the availability of the resources and minimizes the waiting time in the local queues of the clusters, unlike the other scheduling algorithms that try to find the best resource that corresponds to the job requirements; otherwise, the job will remain in the global queue of the Metascheduler, which implies an increase in the waiting time of the job.

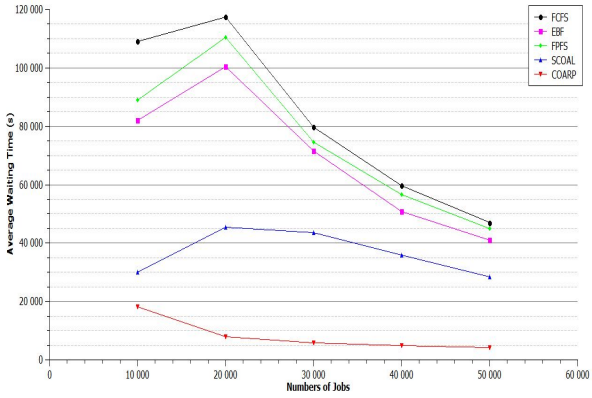

Figure 5: average waiting time of all the submitted the jobs

Resources utilization: From Figure 6 we can see that the utilization rate of the resources remains stable in time and by comparing the five curves of Figure 6, we see that our co-allocation strategy give us a maximization of resources utilization rate compared to the other scheduling algorithms. This is due to the co-allocation strategy that allows the fragmentation of the job on the most available resources when there is no appropriate resource, unlike the other scheduling algorithms that try to find the best resource that corresponds to the job requirements; otherwise, the job will remain in the global queue, which implies an under-utilization of the resources.

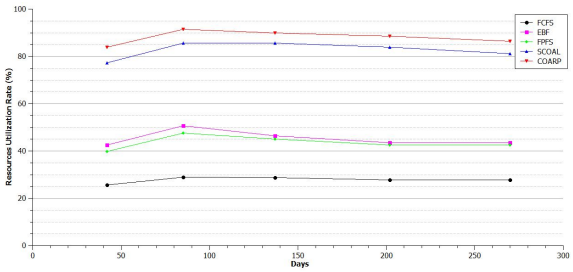

Figure 6: Resources utilization rate

System Load Balancing: From Figure 7 we can see that the standard deviation remains stable in time and by comparing the five curves of Figure 7, we see that our co-allocation strategy load the balance among the resources providers compared to the other scheduling algorithms. This is due to the offer mechanism that allows the local schedulers to generate offers according to their computing capabilities, unlike the other scheduling algorithms that consult Global information System (GIS) to schedule the jobs. These GISs are often centralized and have an asynchronous view on the global state of the system.

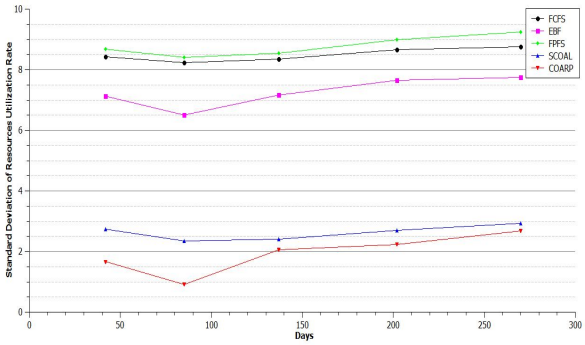

Figure 7: Standard deviation of resources utilization

5 CONCLUSION AND FUTURE WORK

The success of grid computing will depend on the effective utilization of the grid's resources for various computationally jobs. Given a vast number of resources that are available on a Grid, an important problem is the co-allocation of the jobs on the grid with various objectives: Makespan, waiting time, resources utilization rate and load the balance. In this paper, we introduced a co-allocation policy for composing resource offers from multiple resources providers to co-allocate a grid user's jobs. These offers express the interest of resource providers in executing an entire job or only part of it without revealing their local load and total system capabilities. When the Metascheduler receives offers to meet user requirements, it can decide how to submit the job among the resource providers.

We extend the GridSim Tool Kit to carry out the simulation of our co-allocation algorithm to reduce the total time to release user jobs and waiting time in the global queue, maximize the resources utilization rate and load the balance among the resources providers, and compared our results with FCFS, EBF, FPFS and simple co-allocation algorithms (SCOAL). We draw the conclusion that our co-allocation algorithm presented in this paper is better than algorithms FCFS, EBF, FPFS and simple co-allocation (SCOAL).

In this work, we assume that there is no communications among different jobs or different tasks of a job. Usually, the jobs are independent of each other in the grid, but different tasks of a job may require communicating, hence, it is an interesting direction for future research. In the future, we should also consider some fault tolerant measures to increase the reliability of our algorithm.

REFERENCES

[1] Abramson D., Giddy J., Foster I. "High performance parametric modeling with nimrod/G: killer application for the global grid?", Proceedings of Inter Symp on Parallel and Distributed Processing, Chicago, IEEE Computer Society Press, 2000.

[2] Bal H., Bhoedjang R. R., Hofman R., "The distributed ASCI supercomputer project", ACM Operating Systems Review, 34(4), 76−96, 2000.

[3] Bucur A. I. D., Epema D. H. J., "The performance of processor co-allocation in multicluster systems", Proceedings of IEEE/ACM Inter Symp on Cluster Computing and the Grid, Tokyo, IEEE Computer Society Press, 302−309, 2003.

[4] Bucur A. I. D., Epema D. H. J., "Scheduling policies for processor co-allocation in multicluster system", IEEE Trans on Parallel and Distributed Systems, 18(7), 958−962, 2007.

[5] Buyya R., "Economic-based distributed resource management and scheduling for grid computing", Melbourne, Monash University, 2002.

[6] Buyya R., Abramson D., Venugopal S., "The grid economy", Proceeding of the IEEE, 93(3), 698−714, 2005.

[7] Buyya R., Murshed M., "GridSim : A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", The Journal of Concurrency and Computation, Practice and Experience (CCPE), Volume 14, Issue 13-15, Wiley Press, 2002.

[8] Czajkowski K., Foster I., Kesselman C., "Resource coallocation in computational grids", Proceedings of Inter Symp on High Performance Distributed Computing, California IEEE Computer Society Press, 219−228, 1999.

[9] Feitelson D. G., Rudolph R., "Parallel Job Scheduling: Issues and Approaches", In Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1995 and JSSPP 1995. LNCS, vol. 949, pp. 1–18. Springer, Heidelberg, 1995.

[10] Foster I., Kesselman C., Lee C., et al, "A distributed resource management architecture that supports advance reservation and co-allocation", Proceedings of the 7th International Workshop on Quality of Service (IWQoS'99), Jun 1-4, 1999, London, UK. Los Alamitos, CA, USA, IEEE Computer Society, 27-36, 1999.

[11] Foster I., Kesseleman C., Tuecke S., "The Anatomy of the Grid: Enabling Scalable virtual Organizations", International Journal of Super Computer Applications 15, 3, 2001.

[12] Foster I., Iamnitchia A., "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing", In Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003, LNCS, vol. 2735, Springer, Heidelberg, 2003.

[13] Foster I., Roy A., Sander V., "A quality of service architecture that combines resource reservation and application adaptation", Proceedings of the 8th International Workshop on Quality of Service (IWQoS'00), Jun 5-7, 2000, Pittsburgh, PA, USA. Los Alamitos, CA, USA,IEEE Computer Society, 181-188, 2000.

[14] He L. G., Jarvis S. A., Spooner D. P., "Allocating non-real-time and soft real-time jobs in multiclusters", IEEE Trans on Parallel and Distributed Systems, 17(2), 99−112, 2006.

[15] Herroelen W., De Reyck B., Demeulemeester E, "Resource constrained project scheduling: A survey of recent developments", Computers and Operations Research 25, 279–302, 1998.

[16] Leinberger W., Karypis G., Kumar V., "Job scheduling in the presence of multiple resource requirements", Proceedings of ACM/IEEE Conf on Supercomputing, Portland IEEE Computer Society Press, 1999.

[17] Mohamed H. H., Epema D. H. J., "An evaluation of the close-to-files processor and data co-allocation policy in multiclusters", Proceedings of Inter Conf on Cluster Computing, San Diego IEEE Computer Society Press, 287−298, 2004.

[18] Nisan N., London S., Regev O. "Globally distributed computation over the internet: The POPCORN project", Proceedings of Inter Conf on Distributed Computing Systems, Amsterdam, IEEE Computer Society Press, 592−601, 1998.

[19] Roy A., Sander V., "Advance reservation API", Technical Report GFD-E.5, Scheduling Working Group, Global Grid Forum, 2002.

[20] Snell Q., Clement M., Jackson D., et al, "The performance impact of advance reservation metascheduling", Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP'00), May 1-5, 2000, Cancun, Mexico. Los Alamitos, CA, USA, IEEE Computer Society, 137-153, 2000.

[21] Waldspurger C. A., Hogg T., Huberman B. A., et al, "Spawn: A distributed computational economy", IEEE Trans on Software Engineering, 18(2), 103−117, 1992.

[22] Wolski R., Brevik J., Obertelli G., Spring N., "Writing programs that run EveryWare on the computational grid", IEEE Trans on Parallel and Distributed Systems, 12(10), 1066−1080, 2001.