Resources Co-allocation Strategies in Grid Computing

Sid Ahmed MAKHLOUF
Department of Computer Science
University of Oran, Algeria
Email: sidahmed.makhlouf@gmail.com

Belabbas YAGOUBI
Department of Computer Science
University of Oran, Algeria
Email: 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 the 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. Due to the lack of centralized control and the dynamic nature of resource availability, any successful co-allocation mechanism should be highly distributed and robust to the changes in the Grid environment. Moreover, it is desirable to have a co-allocation mechanism that does not rely on the availability of coherent global information. This work addresses those problems by describing and evaluating two resources co-allocation Strategies. The proposed Strategies have been verified through an extension of *GridSim* simulation toolkit and the simulation results confirm that our Strategies allow us to achieve the most of our goals.**

Keywords: **Grid Computing, Grid Scheduling, Resources Allocation, Resources Co-allocation, Multi Agents Systems, Advance Reservation, Reinforcement Learning**

I. Introduction and Related Work

Grid Computing is concerned with *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations* [8]. The coordination between multiple administrative domain results heterogeneity in Grid Environment. 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.

To take full advantage of the promising capabilities of grid computing, good resources co-allocation schemas must be developed. Compared to the resources allocation in traditional distributed systems, resources co-allocation in the grid must consider the characteristics of users and the nature of resources, making the design of a co-allocation system very complicated [1]. Sometimes, the needs of a single job may exceed the capacity available in each of the subsystems making up a Grid, and so co-allocation [5] (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.

The advance reservation is an effective technique to ensure quality of service. It was incorporated in several grids. It allows applications to obtain simultaneous access to adequate resources and ensure their availability at the required time [6]. Several studies have been proposed using the advance reservation [11], [18]. However, advance reservation have many negative effects on resource utilization rate and jobs scheduling in the Grid systems. Several studies [7], [16] show that the advance reservation reduces the resources utilization rate and excessive reservation requests often result in a high rate of rejections from the resource providers. These negative effects influence the Grid economy [2], where resource providers wish to increase the utilization rate of their resources to obtain maximal profits.

Agents are known to be an appropriate paradigm for modelling complex, open and distributed systems. In [9] the authors discuss the benefits obtained through the application of multi-agent systems for Grid computing and show that multi-agent systems are well suited to describe the grids, because the distributed nature of autonomous agents (users and resource providers) reflects the nature of federated grid. Therefore, many systems applying multi-agent paradigm for Grid computing have been proposed. In [10], [19] the authors show that the multi-agent systems can effectively solve the problems of load balancing and resource allocation in grid computing. In [4], authors introduce an ontology based on the paradigm of an intelligent agents for resource allocation in the grid, the proposed approach uses the ontological argument to select an appropriate resource provider.

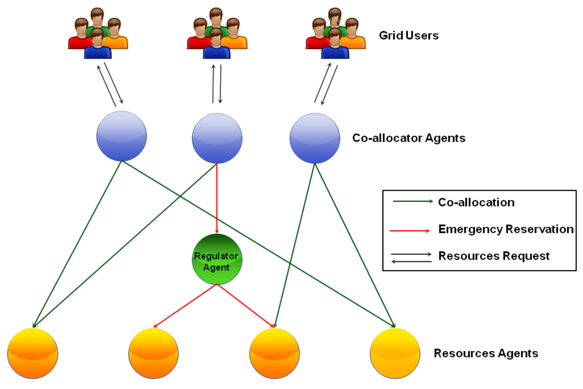
This paper presents agent-based resources co-allocation in Grid computing. A new co-allocation model had been introduced where a set of *co-allocators agents* receive jobs from multiple Grid users and use some techniques to schedule the job to one or more *resources agent* (see section III). The objectives of introducing these co-allocation strategies are as follows: (i) improve users benefit by minimizing their job's execution time and waiting time; (ii) improve resources benefit by maximizing theirs utilization rate. We assume that the co-allocation model is non-preemptive, and all the jobs are independent.

The remaining part of this paper is organized as follows. Our Grid Co-allocation Architecture and Motivations are presented in Section II. In Section III, the algorithmic description of our co-allocation strategies is presented. An experimental setup along with the comparative results is explained in Section IV. Conclusion and future research direction are proposed in Section V.

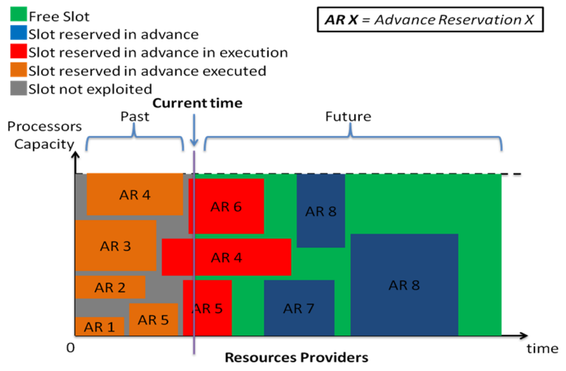
II. System Architecture and Motivations

A. Architecture

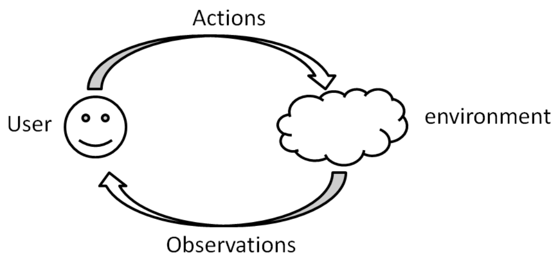
We model the Grid as a heterogeneous system consisting of three different types of agents [15] : *co-allocator agents*, *resources agents* and the *regulator agent* (see Figures 1(a)):



(a) Co-allocation system architecture



(b) Resources provider's slots representation in the time



(c) Reinforcement learning mecanisme

Fig. 1. System Model

Resources Agents**:** is a set of agents that offer resources to serve computational needs of Grid users. Each *resource agent* is characterized by the type and amount of the resources under her disposal, support advance reservation and represents one or more resources providers. Each resource provider may differ from the rest of the others ones with respect to number of processors, speed of processing, local scheduling. The introduction of this type of agent is was necessity because the most of the providers use local resources management that does not support the advance reservation.

Co-allocator Agents **(or** ***Brokers*****):** is a set of "selfish" agents who try to increase their profits. They are interested to reduce at least the time-out and the response time of the applications that they receive.

Regulator Agent**:** it intervenes to solve conflicts between the two or more *co-allocators agents*.

B. Motivations

The lack of precise information on the resources status in large scale is one of the challenges of resources co-allocation in grid computing. The co-allocation systems integrated into the grids do not support a co-allocation in real time as they often retrieve systems information asynchronously. This suggests that the mechanisms for resources co-allocation should not depend on the availability of global information.

Conventionally the advance reservation is defined as a process of allocating resources for use at a specific time in the future [12]. The two main attributes of a reservation request is the reservation start time and the reservation duration time (Figures 1(b)). As the availability and performance of resources are unpredictable in the grid systems on a large scale, the estimation of these two attributes are difficult. Therefore, the advance reservation has many disadvantages on the resources sharing and scheduling applications in grids.

In many problems, it is desirable that a user of the Grid can reach a goal without knowing the rules to achieve the goal in question [17]. The Reinforcement Learning is a method to learn dynamically these rules by interacting with its environment. The interaction between the user and its environment is permanent (Figures 1(c)). The user chooses to make *action* and the environment changes according to the action, so, the user will be confronted to choose new actions. The user does not have the complete perception of his environment, this restriction imposes an additional difficulty on decision-making. A particularity appropriate to the reinforcement learning is the use of a *reward* after each action. The reward is an integer sent by the environment to the user for each action carried out. The goal of the user is to increase the total sum of the received rewards. The reinforcement learning allows a system to adapt itself to the environment changes, such as change of the resources capacities, resources insufficiency, or arrived of the users in the system.

The MAS *(Multi Agents System*) approach is well suited for describing the Grid, because the distributed, autonomous nature of agents (*Grid users and resources*) reflects the federated nature of the Grid. Introducing resources offers allows the multi agents system to adapt to changes, such as the changing resource capacities, resource failure, or introduction new agents into the system.

Motivated by these facts, we propose two distributed resources co-allocation strategies, the first strategy use resources offers and advance reservation planning (*OAR*) and the second use the reinforcement learning mechanism (*RLA*). The objective of *OAR* and *RLA* is to minimize the user's jobs response time, minimize the waiting time of the users and increase the resource utilization rate.

III. Co-allocation Model

Proposal co-allocation system supports *Bag of Tasks* applications (BoT) where they are mainly for intensive calculation. A *broker* (*co-allocator agent*) can decompose a *BoT* application into k sub-applications (fragments of application), where every sub-application is sent to a *resources agent* [14].

$J_{(W,E)}^{(k,P,L)}$: The $k-th$ application submitted to a *resources agent*. It needs P processors and has a size of L MI (Million Instructions). W and E are respectively the waiting and the execution time of the application. At the submission time $W = 0$ and $E = 0$.

O_c : Offer from the *resource agent c*, it consists of a list of slots. A slot $S_{(i,c,D)}^{(s,n)}$ is windows for a *resource agent c* that represent it's start time availability s. Each slot i consist of a number available processors n at time s and duration time D to execute a job. D is calculated by the *resources agent c* by using the resources providers processing *speed* and L witch is the size of the received application. We can represent an offer from *resource agent c* as $O_c = \left\{ S_{(i,c,D)}^{(s,n)} | i = \overline{1, |O_c|} \right\}$. Offers are to execute part or the entire job.

A. Resources Agents

The *resource agent :* it's goal is to increase its resources utilisation rate by giving the same offer to several *co-allocators agents*. The *resource agent* uses the job information provided by the *co-allocator agent*. In order to generate an offer O_c for a job $J_{(W,E)}^{(k,P,L)}$, the *resource agent c*:

1) Calculate the execution duration time D of the job;
2) Find all available free slots in the advance reservation list;
3) Create a list of free slots O_c for each job $J_{(W,E)}^{(k,P,L)}$. Each 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 agent* identifier c;
4) Return the offer O_c to the *co-allocator agent*.

B. Co-allocator Agents

We developed the *co-allocators agents* to support the strategies of the *advance reservations planning* and the *reinforcement learning* to measure their impacts on the behaviour of the grid.

1) Advance reservations planning strategy: In this strategy, each *co-allocator agent* is responsible for the composition of offers from different *resource agents* to meet the needs of users in terms of resources. The composition determines the workload that each *co-allocator agents* will sent for each *resource agent*. The aim of the offers composition is to minimize the execution and waiting time of jobs user. The offer composition is described in the algorithm 1.

Algorithm 1 Co-allocation algorithm integrated in each *broker* that use Advance reservations planning strategy

```
Require: $J_{(W,E)}^{(k,P,L)}$, $N$
 1: $OffersList \leftarrow \phi$
 2: for $c = 1$ ; $c \leq N$ ; $c \leftarrow c + 1$ do
 3:     $O_c \leftarrow queryFreeSlot\,(c)$
 4:     $OffersList \leftarrow OffersList \cup O_c$
 5: end for
 6: $SortedOffersList \leftarrow sort(OffersList)_{FinishTime}$
 7: $BestOffersList \leftarrow \phi$; $NumPE \leftarrow 0$
 8: while $NumPE < P$ do
 9:     $S_{(i,c,D)}^{(s,n)} \leftarrow \min\,(SortedOffersList)_{FinishTime}$
10:     $BestOffers \leftarrow BestOffers \cup \left\{ S_{(i,c,D)}^{(s,n)} \right\}$
11:     $NumPE \leftarrow NumPE + n$
12:     $SortedOffersList \leftarrow SortedOffersList - \left\{ S_{(i,c,D)}^{(s,n)} \right\}$
13: end while
14: while $BestOffersList \neq \phi$ do
15:     $S_{(i,c,D)}^{(s,n)} \leftarrow get\,(BestOffersList)$
16:     $BestOffersList \leftarrow BestOfferList - \left\{ S_{(i,c,D)}^{(s,n)} \right\}$
17:     if $sendAR\left(s, n, D, J_{(W,E)}^{(k,P,L)}, c\right)$ = failed then
18:         $requestRegAgt\left(s, n, D, J_{(W,E)}^{(k,P,L)}, ThisAgentID\right)$
19:     end if
20: end while
```

After collecting the offers from all the N *resource agents*, the *co-allocator agent*:

1) Creates a $OffersList = \bigcup_{c=1}^{N} O_c$ with all the proposed offers, each offer O_c from *resource agent c* contain a set of free slots $O_c = \left\{ S_{(i,c,D)}^{(s,n)} | i = \overline{1, |O_c|} \right\}$;
2) Sorts the $OffersList$ in ascending order by finish time with *FinishTime = StartTime + DurationTime*;
3) Create a $BestOffersList \subseteq OffersList$ that contains the first best slots in the $OffersList$ according to the job requirement P;
4) For each slot $S_{(i,c,D)}^{(s,n)}$ in the $BestOffersList$ with $i = \overline{1, |BestOffersList|}$, send an advance reservation request $AR\langle s, n, D, J_{(W,E)}^{(k,P,L)}, c\rangle$ to the *resource agent c* that include slot's start time s, slot's available processors n , the job execution duration time D proposed by the *resource agent c* and the job request $J_{(W,E)}^{(k,P,L)}$. If the co-allocator agent receives an error message then that means the slot had been taken by another *co-allocator agent*. This behavior is normal since a *resource agent* can propose the same offer to several *co-allocators agents* to increase its resource utilization rate. In this case, the *co-allocator agent* asks the *regulator agent* to find him a new *emergency reservation* for the sub-job $J_{(W,E)}^{(k,n,L)}$.

2) Reinforcement learning strategy: The *brokers* (*co-allocators agents*) have no preliminary knowledge on the capacities of the *resources agents*, they use the reinforcement learning mechanism to estimate their capacities by using their experiences gained in the past. So to do this, they attribute a *score* for each *resources agent*. This *score* represents an evaluation of the performances of the resources agent. He indicates how this last one behaves in past. After the execution of each application, the broker updates the *score* of the corresponding *resources agent* [13].

When the *broker* a successfully receives the execution result of the application $J_{(W,E)}^{(k,P,L)}$, it updates the score R_c^a of the *resources agent* c that have finished the execution in question, so to do this:

1) It calculates the response time $T_c = W + E$ for the *resources agent* c.
2) It calculates the average response time $T_c^w \leftarrow \frac{T_c^w + T_c}{k}$ of the k applications submitted to the *resources agent* c.
3) It calculates the reward $r_c = |T_c^w - T_c|$ of the *resources agent* c.
4) It updates the score $R_c^a \leftarrow R_c^a + r_c$ of the *resources agent* c.

As indicated below, every *broker* a uses his experience to distribute an application between the N *resources agents* with $1 \leq a \leq M$ Where M is the number of *brokers* in the system. For each *resource agent* c the *broker* a keeps a score R_c^a to indicate him his efficiency in past and to calculate him the size of the fragment of the application which he is going to run. The process of co-allocation integrated into each *broker* is described in the algorithm 2 for each new received application.

1) The *broker* a calculates R^T which is the sum of the scores of all the *resources agents*.
2) The *broker* a calculates F_c which is the size of the fragment of the application that each *resource agent* will take. More the resource agent has an important score more the size of the fragment which he is going to receive is important.
3) The *broker* a sends each fragment of the application $J_{(w,e)}^{(k,F_c,L)}$ to the *resource agent* corresponding by calling the function $sendFragment\,()$. If the query fail then it means that the *broker* in question overestimate the capacities of the *resources agent*. In this case, the *broker* in question asks to the *regulator agent* to find him another reservation by calling the function $requestRegAgt\,()$.

Algorithm 2 Co-allocation algorithm integrated in each *broker* that use reinforcement learning strategy

```
Require: $J_{(W,E)}^{(k,P,L)}$, $N$, $\{R_1^a, R_2^a, ..., R_c^a, ..., R_{N-1}^a, R_N^a\}$
 1: $W \leftarrow 0$; $E \leftarrow 0$; $R^T \leftarrow 0$
 2: for $c \leftarrow 1$ ; $c \leq N$ ; $c \leftarrow c+1$ do
 3:     $R^T \leftarrow R^T + R_c^a$
 4: end for
 5: for $c \leftarrow 1$ ; $c \leq N$ ; $c \leftarrow c+1$ do
 6:     $F_c \leftarrow \frac{1}{R^T} * (R_c^a * P)$
 7: end for
 8: for $c \leftarrow 1$ ; $c \leq N$ ; $c \leftarrow c+1$ do
 9:     if $sendFragment\left(c, J_{(W,E)}^{(k,F_c,L)}\right)$ = failed then
10:         $requestRegAgt\left(J_{(W,E)}^{(k,F_c,L)}\right)$
11:     end if
12: end for
```

This policy of co-allocation distribute an application on all the resources agents of the system.

C. Regulator Agent

The *regulator agent* is responsible for requesting *emergency offers* from the different *resource agents* to meet a sub-jobs requirement that was sended by the *brokers*. The goal of the *regulator agent* is to find the best offer so that the system remains always stable in terms of execution time and waiting time after the appearance of a conflict.

After collecting the offers from all the N *resource agents* for the sub-job $J_{(W,E)}^{(k,n,L)}$ that require n processors and has length of L MI (*Million Instructions*), the *regulator agent*:

1) Creates a reservation list $RvList = \bigcup_{c=1}^{N} Rv_c^{(s,D)}$ with all the proposed reservations, each reservation $Rv_c^{(s,D)}$ from *resource agent* c has a start time s and duration time D;
2) Take the best advance reservation $BestRv_c^{(s,D)} \subset RvList$ that has the earliest finish time with $FinishTime = s + D$;
3) Cancel the rest of the reservations in the $RvList$;
4) For the reservation $BestRv_c^{(s,D)}$ send a commit message $CAR\langle s, D, J_{(W,E)}^{(k,n,L)}, c\rangle$ to the *resource agent* c that include reservation's start time s, the job execution duration time D proposed by the *resource agent* c and the sub-job request $J_{(W,E)}^{(k,n,L)}$;

IV. Experiment Study

A. Experimental Setup

We have evaluated our co-allocation policies by means of simulations to observe its effect in a long-term usage. We have used the event-driven simulator named *GridSim* [3], which we have extended to support jobs on multi-site environments and MAS (*Multi Agent System*). We have used real traces from supercomputers available at the *Parallel Workloads Archive*[1]. In our simulation, various entities connected by a network and every two entities' connectivity had an exclusive bandwidth.

We analyse the global behaviour of the system using *OAR* strategy that use resources offers and advance reservation planning and compare it with the strategy *RLA* that makes use of a agent knowledge of current resource usages. We have modelled an environment composed of seven *resources agents*, each *resources agent* manage one cluster with it own local scheduler, one *regulator agent* and seven *co-allocators agents* that receives jobs, Table I illustrate our grid environment. We have used the trace file of the *San Diego Supercomputer Center Blue Horizon*. More details on the trace file can be found at the Parallel Workloads Archive. We have simulated 503 days of this traces that is equivalent to 117000 jobs.

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

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

TABLE I
COMPUTATIONAL GRID ENVIRONMENT

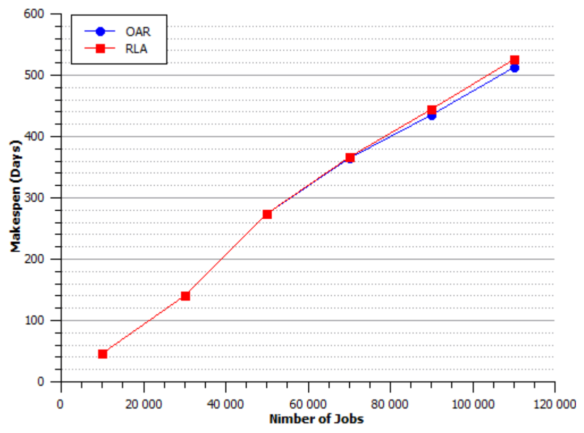
Resource Provider	*Processors Numbers*	*Resource Speed* (MIPS)
Cluster 1	240	1000
Cluster 2	210	1100
Cluster 3	180	1200
Cluster 4	150	1300
Cluster 5	120	1400
Cluster 6	90	1500
Cluster 7	60	1600

- **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;
- **Number of the *regulator agent* interventions:** We count the number of times $C = \sum_{i=1}^{n} C_i$ where the *co-allocator agents* call the *regulator agent* with n is the number of the *co-allocator agents* and C_i is the number of calls to the *regulator agent* of the *co-allocator agent* i.

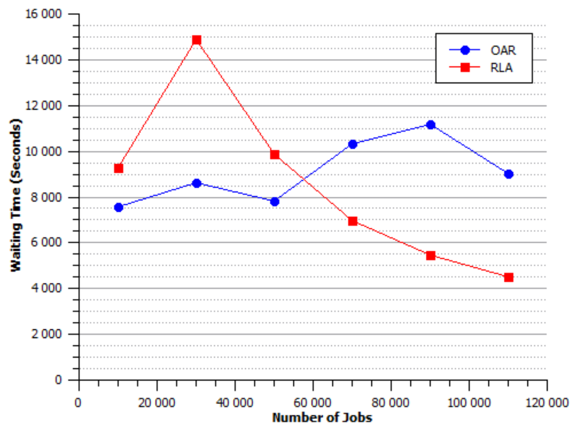
B. Results analysts

1) User performance: From the Figures 2(a) we can see that when the number of jobs increases the *Makespan* increases and by comparing the two curves of the Figures 2(a), we see that we have obtained a gain in *Makespan* in large scale by using our *OAR* algorithm compared to the *RLA* algorithm. This is due to the *Demands/Offers* strategy that allows the distribution of the job's execution time on the *most available resources*, unlike the *RLA* algorithm that try to *estimates the best resources* that can satisfy the job's requirements.

From the Figures 2(b) we can see that when the number of jobs increases the average waiting time decreases and by comparing the two curves of the Figures 2(b), we see that in the large scale the *RLA* algorithm reduce the average waiting time of the jobs compared to the *OAR* algorithm. This is due to the *reinforcement learning* mechanism that allows to the *co-allocator agents* to have a global vision of the system via the response time of the returned jobs, in this algorithm the *co-allocators agents* try to send the jobs to the *resources agents* that have the best score in terms of response time, unlike the *OAR* algorithm that uses the *advance reservation* mechanism to ensure the availability of the resources in the future.



(a) Total Makespan to execute all the jobs

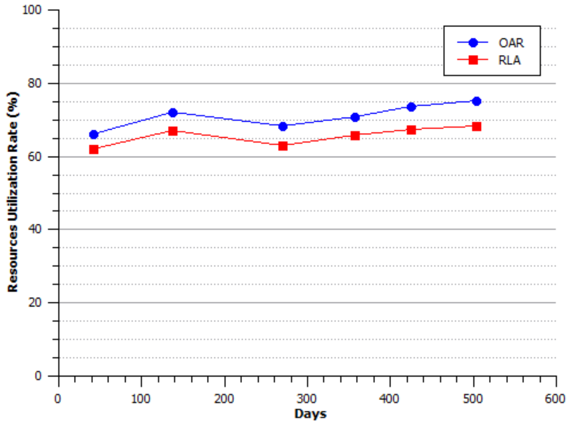


(b) average waiting time of all the submitted the jobs

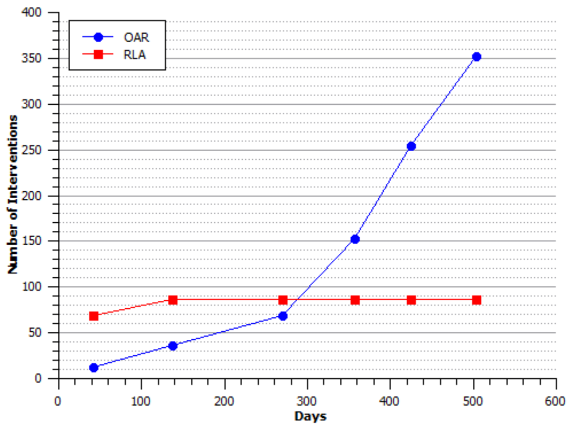
Fig. 2. User performance

2) System performance: From the Figures 3(a) we can see that the utilization rate of the resources remains stable in time and by comparing the two curves of the Figures 3(a), we see that the *OAR* strategy give us a maximization of resources utilization rate compared to the *RLA* algorithm. This is due to the *Demands/Offers* policy that allows to a *resource agent* to maximize the use of its resources by giving the same offer to several *co-allocator agents*, unlike the *RLA* algorithm that use a reinforcement learning techniques to schedule jobs, these techniques do not allow to the *resources agents* to maximize the use of their resources.

By comparing the two curves of the Figures 3(b), we see that the *RLA* algorithm remains stable in time and the *OAR* increase in time, this is due to the *reinforcement learning* techniques that allow to the *co-allocator agents* to know the global stat of the system after few time, and by consequence, the *co-allocator agents* do not need the help of the *regulator agent*, unlike the *OAR* algorithm that use the offers concurrence to schedule the jobs and by consequence, there is a risk to have a offers conflicts.



(a) Resources utilization rate



(b) Number of the regulator agent interventions

Fig. 3. System performance

V. Conclusion and Future Work

In this paper, we introduced a MAS based co-allocation policy for composing resource offers from multiple resources providers to co-allocate a grid user's jobs. so, we introduced 3 types of agents: *Co-allocator agents*, *resources agents* and *regulator agent*. We extend the *GridSim* Toolkit to carry out the simulation of our co-allocation policy and compared our results with an extension of a distributed resource allocation that use reinforcement learning agents (*RLA*). We draw the conclusion that regarding to the execution time and utilization rate, our policy gives us good performances compared to the *RLA* policy, but regarding to the waiting time and the number of conflicts in the system, the *RLA* policy gives us good performances compared to our policy. We remark that the two policy (*OAR* and *RLA*) have advantages and disadvantages, so as future work we think to develop a new hybrid model that includes the two models together to profit from their advantages.

References

[1] H. Blanco, J. Lrida, F. Cores, and F. Guirado, "Multiple job co-allocation strategy for heterogeneous multi-cluster systems based on linear programming," *The Journal of Supercomputing*, pp. 1–9, 2011.

[2] R. Buyya, D. Abramson, and S. Venugopal, "The Grid Economy," in *Proceedings of the IEEE*. IEEE Press, New Jersey, USA, 2005, vol. 93, no. 3, pp. 698–714.

[3] R. Buyya and M. M. Murshed, "Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[4] K. C. Cho, C. H. Noh, and J. S. Lee, "Ontology-based intelligent agent for grid resource management," in *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems, First International Conference, ICCCI 2009, Wroclaw, Poland, October 5-7, 2009. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5796. Springer, 2009, pp. 553–564.

[5] K. Czajkowski, I. Foster, and C. Kesselman, "Resource co-allocation in computational grids," in *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society, 1999, p. 37.

[6] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in *Proceedings of the International Workshop on Quality of Service*, 1999, pp. 27–36.

[7] I. Foster, A. Roy, V. Sander, and F. J. Gmbh, "A quality of service architecture that combines resource reservation and application adaptation," Tech. Rep., 2000.

[8] I. T. Foster, "The anatomy of the grid: Enabling scalable virtual organizations," in *Proc. First IEEE International Symposium on Cluster Computing and the Grid (1st CCGRID'01)*. IEEE Computer Society (Los Alamitos, CA), 2001, pp. 6–7.

[9] I. T. Foster, N. R. Jennings, and C. Kesselman, "Brain meets brawn: Why grid and agents need each other," in *AAMAS*. IEEE Computer Society, 2004, pp. 8–15.

[10] M. Meriem and Y. Belabbas, "Dynamic dependent tasks assignment for grid computing," in *Algorithms and Architectures for Parallel Processing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6082, pp. 112–120.

[11] C. Qu, "A grid advance reservation framework for co-allocation and co-reservation across heterogeneous local resource management systems," in *Parallel Processing and Applied Mathematics, 7th International Conference (7th PPAM'07)*, ser. Lecture Notes in Computer Science (LNCS), vol. 4967. Springer-Verlag (New York), 2008, pp. 770–779.

[12] A. Roy and V. Sander, "Advance reservation api," Global Grid Forum (GGF), 2002.

[13] M. Sid Ahmed and Y. Belabbas, "Co-allocation des ressources dans les grilles de calcul utilisant le renforcement de l'apprentissage," in *Les premières journées doctoriales en informatique (ReSyD2010)*. BEJAIA University, 2010.

[14] ——, "Co-allocation in grid computing using resources offers and advance reservation planning," in *International Symposium on Modelling and Implementation of Complex Systems Constantine (MISC'2010)*. UMC University, 2010, pp. 108–117.

[15] ——, "Distributed resources co-allocation in grid computing," in *International Conference on Machine and Web Intelligence (ICMWI2010)*. USTHB University, 2010, pp. 223–228.

[16] Q. Snell, M. Clement, D. Jackson, and C. Gregory, "The performance impact of advance reservation meta-scheduling," *Lecture Notes in Computer Science*, vol. 1911, pp. 137–153, 2000.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[18] A. Takefusa, H. Nakada, T. Kudoh, and Y. Tanaka, "An advance reservation-based co-allocation algorithm for distributed computers and network bandwidth on qos-guaranteed grids," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6253, pp. 16–34.

[19] B. Yahaya, R. Latip, M. Othman, and A. Abdullah, "Dynamic load balancing policy in grid computing with multi-agent system integration," in *Software Engineering and Computer Systems*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2011, vol. 179, pp. 416–424.