A New Randomized Algorithm for Handling Scheduling Conflicts in Grids

Hamed VAHDAT-NEJAD, Kamran ZAMANIFAR
Computer Engineering Department, University of Isfahan, IRAN
vahdatnejad@eng.ui.ac.ir

***Abstract*—As the scale of Grid platforms grows, the idea of a centralized scheduler loses its efficiency, and it is replaced with the scheme of decentralized schedulers. However, a new problem emerges in distributed scheduling systems, which is how to coordinate the autonomous schedulers to avoid the occurrence of conflicting schedules. In this paper, by exploiting the idea of randomized algorithms, a new scheduling scheme has been proposed, which addresses the problem of scheduling conflicts. The proposed algorithm is thoroughly decentralized in the sense that there is no central point of contact in the system. In addition, our approach is a suitable way toward reaching scalability and autonomy in future Grids. We prove the feasibility and effectiveness of the proposed algorithm through statistical analysis.**

***Index Terms*—Coordination, Distributed system, Grid, Randomized Algorithms, Scheduling**

I. INTRODUCTION

A grid computing infrastructure [1] is a collection of computational, storage and other resources connected by a network. The grid middleware facilitates the interaction of resources in the grid, and forms the image of a single huge environment. Grid users run their applications on top of the grid middleware layer. A grid environment can execute numerous such applications concurrently.

The grid resource management system (GRMS) controls the exploitation of resources across the grid to reach the goal of a grid system with high performance. Scheduling is the main part of GRMS, which utilizes both the grid and applications information to produce an assignment from user jobs to grid machines [4]. Different scheduling algorithms try to optimize various parameters like job makespan [2, 5] and waiting time, resource utilization, and system throughput. However, scheduling is a complicated problem, which has been shown to be NP complete [3].

One of the characteristics of a good scheduling algorithm for grids is its scalability. Hence, centralized algorithms lose their efficiency, and we need the implementation of decentralized or distributed schemes [2, 10]. On the other hand, distributed scheduling algorithms introduce a new challenge, which is the need for coordination between distributed schedulers to prevent the occurrence of conflicting schedules [8, 9]. For example, consider the grid environment shown in Figure 1, which has *n* number of resources and *m* number of schedulers. Grid users submit their applications to one of the schedulers. These schedulers generate schedules based on the resource information attained from the grid information system (GIS). However, if two schedulers query the GIS at the same time, they will obtain similar information about free resources. Based on this information, they will produce the same mapping of tasks in their applications to the available resources, and hence, conflicting schedules will occur. Therefore, both the grid system and applications of users will suffer from degraded performance.

For better illustration, we assume there are two schedulers S_1 and S_2, and three available machines known as P_1, P_2, and P_3. The tasks J_1 and J_2 are submitted in approximately the same time to the schedulers S_1, and S_2, respectively. The execution time of J_1 in P_1, P_2, and P_3 is 10, 6, and 4, and the execution time of J_2 in P_1, P_2, and P_3 is 17, 11, and 20, respectively. Now, if both the schedulers assign their task to the machine which is expected to have the minimum execution time, the machine P_2 will receive the two tasks sequentially, and the makespan will be equal to 17. On the other hand, if one of the schedulers, (e.g. S_2) assigns the task to P_2, and the other allocates another machine (e.g. P_1) to its task, the makespan metric will decrease to 11.

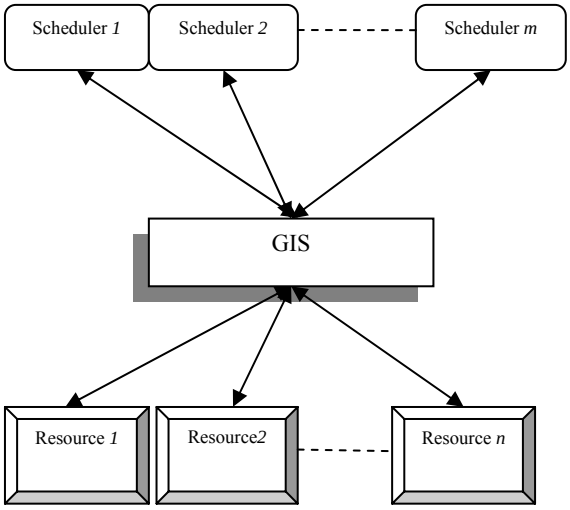

Figure 1. An existing decentralized scheduling approach.

Some of the decentralized schedulers exploit message passing mechanisms to handle this situation. In other words, after determining resources to assign a job, they make them aware and obtain permission of sending the job through messages [2]. However, due to the uncertainty and rigorous dynamicity of the grid environment, and the fact that an entity is not exactly aware of other resources whether being available, free or busy, this mechanism is not efficient. Furthermore, it causes to waste time by sending and receiving messages. On the other hand, in many of real economic grid environments, resources gain money for

Digital Object Identifier 10.4316/AECE.2009.03005

executing jobs or offering services, therefore it is obvious that each resource tries to acquire more jobs to increase its profit.

In [9] a conflict-free resource allocation mechanism has been presented. Its goal is to propose a market mechanism that allows resources from a central resource pool to be allocated to distributed decision makers (agents) that seek to optimize their respective scheduling goals. It assumes that the central pool of resources is owned by a central server (who is the auctioneer), and each job list is represented by a bidder agent who bids for resources to service all the jobs in its job list [9]. The approach is based on the tatonnement [11] process which solves resource conflicts by iteratively updating the prices of resources based on excess demand. The tatonnement process was originally proposed by Walras in 1954 and improved by many other studies [12]. Within each iteration, the auctioneer collects bids from all agents and updates prices in order to solve resource conflicts. A proper design of the price adjustment strategy is essential for fast convergence, since the tatonnement process is known to converge slowly.

In [8], authors tackle the problem of conflicting schedules in workflow scheduling systems. They propose a decentralized and cooperative workflow scheduling scheme, which utilizes a Peer-to-Peer (P2P) coordination space with respect to coordinating the application schedules among the grid wide distributed workflow brokers. However, the approach has problems with scalability, and makes the scheduling process longer.

In this paper, we exploit the idea of randomized algorithms to solve the conflict problem that arises in the decentralized scheduling algorithms. A randomized algorithm is an algorithm that exploits the probability and randomness to solve a difficult problem. These algorithms are one of the significant tools to getting around NP-completeness. There are two main advantages of randomized algorithms: performance and simplicity [7]. Randomized algorithms run faster than the best-known deterministic algorithms. Furthermore, many of them are simpler to describe and implement than deterministic algorithms of comparable performance. The proposed randomized decentralized scheduling (RDS) algorithm is based on the availability of numerous resources in a real grid and is suitable and efficient in highly dynamic environments.

The rest of the paper is organized as follows: in section 2, the grid and scheduling model is presented. Section 3 describes the proposed randomized decentralized scheduling scheme. We evaluate the RDS through statistical analysis in section 4 and conclude the paper in section 5.

System model

The exploited grid model is based on the Grid federation [6] model. We assume a heterogeneous computational network, with numerous available resources. It means that, at any time, there are free resources for executing user jobs in the grid. The assumed grid consists of n resources $\{m_1, m_2,\ldots,m_n\}$ and q schedulers $\{s_1,s_2,\ldots,s_q\}$, which situated in a decentralized manner. The job arrival time to each scheduler follows the Poisson distribution. λ_i is the Poisson rate of submitting jobs to scheduler s_i. To consider grid heterogeneity, jobs and processors are divided into two categories: type 1 and type 2. Only resources with processors of type 1 can execute jobs of type 1. The same statement is true for resources and jobs of type 2.

II. RDS: Randomized Decentralized Scheduling scheme

Most of the scheduling algorithms are user centric, in that trying to decrease the job completion time, response time or makespan. Therefore, free resources with highest computational potential are the first choices of these schedulers for allocating to the submitted jobs. Figure 2 shows the general algorithm for most of the existing schedulers.

This algorithm assigns the job i to the resource which optimizes a metric. Most of the schedulers follow this algorithm, and only differentiate in the metric function. Usually this function is selected so as to minimize the overall completion time of jobs. In other words, it tries to optimize the makespan metric. For example, this function is the minimum function for the online Min-Min algorithm.

(1) For each submitted job i do
(2) For each free resource j with the potential to execute i do
(3) Compute $CT_{i,j}$ = *Completion Time*(job *i*, resource *j*)
(4) End for
(5) Compute *metric(i)* = $f(CT_{i,1}, CT_{i,2}, \ldots\ldots)$
(6) Assign job i to the resource which is corresponded to metric(i)
(7) End for

Figure 2. The General Scheduling Algorithm.

```
(1)  For each submitted job i do
(2)      For each free resource j with the potential to execute i do
(3)          Compute CT_{i,j} = Completion Time(job i, resource j)
(4)      End for
(5)      For counter = 1 to k    do
(6)          Compute metric(i) = f (CT_{i,1}, CT_{i,2}, ……)
(7)          Insert the resource which is corresponded to metric(i) to
             the list of potential resources for executing the job i.
(8)          Remove this resource from the list of available resources.
(9)          If there is not any available resource with the capability
             to execute the job, break
(10)     End for
(11)     Select one of the resources in the potential list randomly, and
         Assign the job to it.
(12) End for
```

Figure 3. The randomized scheduling algorithm.

As stated before, the problem of these scheduling algorithms is more obvious in distributed and decentralized environments. For example, if two jobs of type 1 are submitted to two schedulers at approximately the same time, each of these jobs is only executable on resources of type 1, it is almost certain that the result of both of the schedulers will be the resource which has the most capability among all resources of type 1. Consequently, both of the jobs will be sent to that resource (scheduling conflict) and will be executed sequentially. Therefore, the grid performance will be degraded and job's makespan will be increased.

For solving this problem, we introduce the RDS algorithm, which is based on the randomized algorithms subject. The main assumption of the RDS algorithm is the existence of numerous available resources and hosts in the grid. Figure 3 shows the general form of the randomized scheduling algorithm.

The randomized scheduling algorithm is composed of two phases. In the first phase, a list of maximum k elements of potential resources for executing the job is obtained. This list is created by using the metric that exists in many of the scheduling algorithms. This is done through lines 5 to 10. Afterward, in the second phase we assume the resources in the potential list have uniform distribution, select one of them randomly and assign the job to it. Line 11 performs this. Since we did not make any restriction on the metric function, many of the scheduling algorithms can be moulded in this model; hence we can construct their randomized version, which has better performance in dynamic decentralized environments.

III. EVALUATION

In this section, we compare the scheduling conflicts in generalized decentralized schedulers and RDS. First, we assess the probability of occurring scheduling conflicts in generalized decentralized schedulers, and then, this probability is computed for RDS algorithm.

A. Scheduling conflicts in decentralized schedulers

We assume the probability for a job to be of kind 1 is p_2, and the probability for a resource to be able to execute a job of kind 1 is p_1. Hence, the probability for a job to be of kind 2 is 1-p_2, and the probability for a resource to be able to execute a job of kind 2 is 1-p_1. Furthermore T_{Sij} is the time interval between sending a job *A* from scheduler S_i to the resource m_j and when the update message of resource m_j reaches to other schedulers. We want to compute the probability of sending a job B by another scheduler s_l to the resource m_j in this time interval. By the implicit assumption that the resource m_j has the highest computational capability between free resources which are able to execute the job A (sent by scheduler s_i), this probability equals:

$$\text{Prob}(L) = \text{P(a job B is submitted to } s_l \text{ in this time -interval)} \times \text{P(The job B can be executed on } m_j\text{)} \quad (1)$$

By analyzing the assumption that λ_i is the Poisson rate of submitting jobs to scheduler s_i , the first quantity of the above expression equals:

$$\text{P (a job B is submitted to } s_l\text{)} = MAX\left\{\frac{T_{S_{ij}}}{\lambda_l}, 1\right\} \quad (2)$$

Furthermore, the probability that the job B is executable on resource m_j, can be computed as the addition of two probabilities. The probability that job B is of type 1 and the jobs of type 1 are executable on resource m_j (equals to $p_2\, p_1$), plus the probability that job B is of type 2 and the jobs of type 2 are executable on resource m_j (equals to $(1-p_2)(1-p_1)$).

Therefore $(\text{assume} T_{S_{ij}} < \lambda_l)$

$$prob(L) = p_2 p_1 \frac{T_{S_{ij}}}{\lambda_l} + (1-p_2)(1-p_1)\frac{T_{S_{ij}}}{\lambda_l} \quad (3)$$

So the probability that no other scheduler sends a job to m_j in this time interval equals to:

$$\text{Prob (no conflict)} = \prod_{\substack{i=1 \\ i \neq i}}^{q} (1 - prob(1)) \quad (4)$$

where q is the number of schedulers.

For better analysis we assume that $p_1 = p_2 = 0.5$, $\lambda_1 = \lambda_2 = \ldots = \lambda_q = \lambda$, and $T_{Sij} = T_s$ for all i and j. Hence, the probability that no other scheduler sends a job to m_j in this time interval equals to:

$$\text{Prob (no conflict)} = \left(1 - \frac{T_s}{2\lambda}\right)^{q-1} \quad (5)$$

This expression represents the probability of the occurrence of no scheduling conflict with a given schedule (schedule of job *A* by scheduler S_i). For better illustration, we plot the diagram of this probability with the assumption of T_s=1 in two phases. This assumption states that the time interval between sending a job from a scheduler to a resource, and when the update message of the resource reaches to other schedulers, is just 1 second. It is an ideal assumption, because in reality, this quantity is usually larger than 1 second. In the first phase, we assume λ=60 seconds, and plot the diagram versus the number of schedulers (q) in Figure 4. In other words, we assume that every 60 seconds a job is submitted to each of the schedulers, averagely. As it can be seen in this plot, when there are more than 20 decentralized schedulers, the probability of absence of conflicts becomes less than 0.8. On the other hand, the probability of the occurrence of a conflict becomes more than 0.2, which causes the increase of makespan. As it is obvious in this figure, the bigger the number of schedulers, the smaller the probability of absence of conflicts; hence the bigger the probability of conflict occurrence.

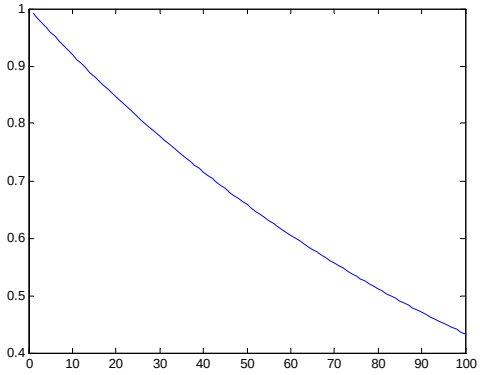

Figure 4. The probability of absence of conflicts versus number of schedulers (λ=60).

In the second stage, we assume that there are 20 decentralized schedulers. Figure 5 shows the diagram of the probability of absence of conflicts versus the Poisson rate of arriving jobs to schedulers. As it can be seen, when λ=50 seconds, this probability is near to 0.8. In other words, when there are 20 decentralized schedulers, and one job is submitted averagely every 50 seconds to each of the schedulers, the probability of conflict occurrence is 0.2. When the Poisson rate decreases, the probability of conflict absence diminishes, which is fully predictable. Because the reduction of λ is equivalent to faster submission of jobs to schedulers, we obtain the increase in the number of scheduling conflicts between decentralized schedulers.

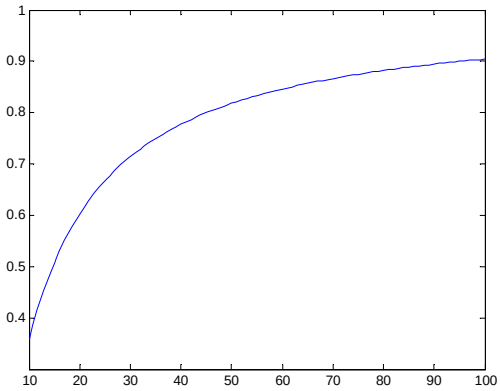

Figure 5. The probability of absence of conflicts versus job arrival rate (q=20).

B. Scheduling conflicts in RDS

Now we discuss the probability of scheduling conflicts in the RDS scheme. We assume there are always k available resources in the potential list. In the RDS scheme, a scheduler sends a job to one of the k potential resources, according to the uniform distribution of resources in list. Regarding the assumptions and reasoning of the previous subsection, here we have Equation 1 as follows

$$\text{Prob(L)} = \frac{1}{k} \times \text{p(a job B is submitted to } s_l \text{ in this time - interval)} \times \text{p(The job B can be executed on } m_j) \quad (6)$$

Hence, the probability of no scheduling conflict occurrence with a given schedule is as follows:

$$\text{Prob (no conflict)} = \left(1 - \frac{T_s}{2k\lambda}\right)^{q-1} \quad (7)$$

The only difference of this equation and equation 5 is the parameter k in the denominator. In fact this coefficient is due to the randomized selection between k potential resources in the RDS. For better analysis we also assume T_s=1 in this case. In other words, we assume that the time interval between sending a job from a scheduler to a resource and when the update message of the resource reaches to other schedulers is just 1 second. In addition, we set the parameter k in the RDS to 10. Figures 6 and 7 correspond to Figures 4 and 5, respectively.

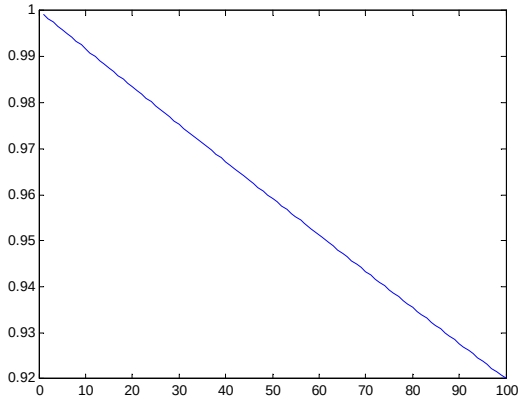

Figure 6. The probability of absence of conflicts versus number of schedulers (λ=60).

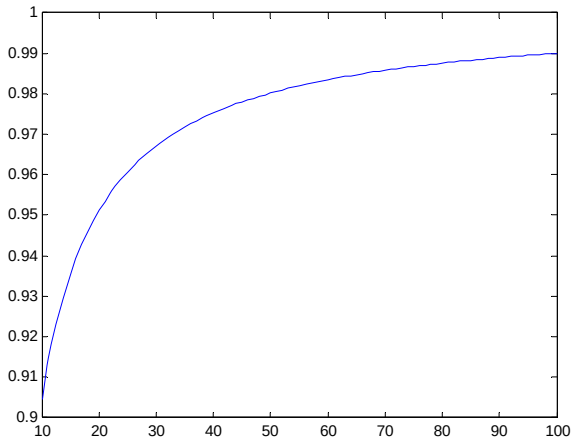

Figure 7. The probability of absence of conflicts versus job arrival rate (q=20)

As it can be seen, in similar situations, the probability of absence of conflicts in RDS raises strongly as compared to the generalized decentralized scheme. In other words, in similar situations, the probability of occurring scheduling conflicts in RDS is smaller than generalized scheme. Therefore, we can expect a much better makespan from RDS. As an example regarding Figures 4 and 6, when the number of schedulers is about 80, the probability of absence of conflicts in the generalized and RDS schemes is 0.5 and 0.93, respectively. Furthermore, the comparison of Figures 5 and 7 reveals the differences between the two algorithms. As an example, when the Poisson rate is about 20 (or averagely every 20 seconds a job is submitted to each of the schedulers), the probability of absence of conflicts in the generalized and RDS schemes is 0.6 and 0.95, respectively. It means that the probabilities of occurring scheduling conflicts in this situation are 0.4 and 0.05 for generalized and RDS schemes, respectively, which shows the differences in avoiding scheduling conflicts between these two approaches.

IV. Conclusion

We have presented a new approach for handling scheduling conflicts in decentralized scheduling schemes. The approach, which is also called RDS, is based on randomized algorithms. Contrary to the coordination algorithms in distributed systems, the RDS imposes no further load to the system. Furthermore, it is very simple and easily implementable. We have shown the efficiency of the RDS through statistical analysis. In the future, we intend to implement the RDS scheme in a decentralized system and compare its efficiency with available coordination schemes in a real Grid environment.

References

[1] I. Foster and C. Kesselman, grid: "Blueprint for a new computing infrastructure", Morgan Kaufmann Publishers, USA, 1998

[2] Hamed Vahdat-Nejad, Reza Monsefi, Hossein Deldari, "Distributed resource scheduling in grid computing using fuzzy approach", Proceedings of the International Conference on Information and Knowledge Technology, Iran, 2007

[3] HU Rong, HU Zhigang, "A scheduling algorithm aimed at time and cost for meta-tasks in grid computing using fuzzy applicability", Proceedings of the eighth International Conference on High-performance Computing in Asia-pacific region, IEEE 2005

[4] Hamed Vahdat-Nejad, Reza Monsefi, Mahmoud Naghibzadeh, "A new fuzzy algorithm for global job scheduling in multiclusters and grids", Proceedings of International Conference on Computational Intelligence for Measurement Systems and Applications, Italy, 2007

[5] Hamed Vahdat-Nejad, Reza Monsefi, "Static parallel job scheduling in computational grids", Proceedings of the International Conference on Computer and Electrical Engineering, Thailand, 2008

[6] Rajiv Ranjan, Aaron Harwood, Rajkumar Buyya, "A case for cooperative and incentive-based federation of distributed clusters", Future generation computer systems, Elsevier, 2007

[7] Ashraf Osman, "Introduction to randomized algorithms", West Virginia University

[8] Rajiv Ranjan, Mustafizur Rahman, and Rajkumar Buyya, "A decentralized and cooperative workflow scheduling algorithm", Proceedings of eighth IEEE International Symposium on Cluster Computing and the Grid, 2008

[9] Hoong Chuin Lau, Shih Fen Cheng, Thin Yin Leong, "Multi-period combinatorial auction mechanism for distributed resource allocation and scheduling", Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2007

[10] Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, Loris Marchal, and Yves Robert, "Centralized versus distributed schedulers for bag-of-tasks applications", IEEE Transactions on Parallel and Distributed Systems, Volume 19, No 5, May 2008

[11] L. Walras, "Elements of pure economics", Allen and Unwin, English translation by William Jaffe 1954 (originally published in 1874)

[12] J. Q. Cheng and M. P. Wellmam, "The WALAS algorithm: A convergent distributed implementation or general equilibrium outcomes", Computational Econ. 12: 1-24, 1998