http://ansinet.com/itj

ITJ

ISSN 1812-5638

# INFORMATION TECHNOLOGY JOURNAL

ANSInet

Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (8): 1166-1170, 2007
ISSN 1812-5638


# QoS Sufferage Heuristic for Independent Task Scheduling in Grid

Ehsan Ullah Munir, Jianzhong Li and Shengfei Shi
School of Computer Science and Technology,
Harbin Institute of Technology, Harbin 150001, China

**Abstract:** Reduction in makespan is a fundamental objective of optimizing task scheduling problems in distributed heterogeneous computing systems. In this study, we propose a new task scheduling heuristic for independent task scheduling in Grid named as QoS Sufferage. Our novel heuristic is based on Sufferage heuristic adapted to include network bandwidth as Quality of Service (QoS) parameter. Numerical simulations are carried out to test our proposed QoS Sufferage heuristic in Grid environment and the results reveal that the proposed QoS Sufferage outperforms other heuristics by achieving a significant reduction in makespan.

**Key words:** Grid computing, independent task scheduling, quality of service, QoS Sufferage heuristic

## INTRODUCTION

Due to advances in wide-area network technologies and low cost of computing resources, Grid computing (Foster and Kesselman, 2001) has established itself as an active research area. One motivation of Grid computing is to aggregate the power of widely distributed resources and provides non-trivial services to users (Foster, 2002). A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities (Foster and Kesselman, 1998). The key technologies that affect the Grid efficiency involve Grid resource allocation and management and task scheduling algorithms. The goal of computational Grid is to utilize all available/free computational resources to overcome difficulties brought about by complicated tasks with enormous computing workloads, so one of the current research problems in Grid task scheduling is to devise new and efficient methods for improving computational efficiency.

After a computational job is designed and realized as a set of tasks, an optimal assignment of these tasks to the processing elements in a given architecture needs to be determined, that is the scheduling problem (El-Rewini *et al.*, 1994; Topucuoglu *et al.*, 2000). In general, tasks can be divided into two groups: Independent and dependent (Braun *et al.*, 1998). Dependent task requires the results of its predecessor tasks while independent task does not require communication with other tasks in same metatask. Metatask can be defined as a collection of independent tasks with no inter-task data dependencies (Braun *et al.*, 2001). The most common objective function of task scheduling problems (both for a Grid and for a non-Grid parallel machine) is reduction in makespan (the total time required to complete metatask).

Over the years, task scheduling, an integrated component of computing, problem has become a well-recognized discipline in Grid and identified as NP complete problem (Sinnen, 2007). The unavailability of an exact solution has led to the proposal of several heuristic algorithms such as Min-min, Max-min, Sufferage (Ibarra and Kim, 1977; Freund *et al.*, 1998; Maheswaran *et al.*, 1999), XSufferage and Weighted Mean Execution Time (Casanova *et al.*, 2000; Jinquan *et al.*, 2005). However, these scheduling approaches do not consider the effects of QoS, which is an extensive concept in task scheduling. To overcome this drawback, QoS guided Min-min (Xiaoshan *et al.*, 2003) with QoS as a factor of algorithm was proposed. But this algorithm has certain limitation as well. Later on Dong *et al.* (2006) proposed a new Grid task scheduling strategy based on QoS priority grouping to eliminate the limitations in QoS guided Min-min algorithm. However the algorithm fails to recognize the humongous scale of Grid as it divides the tasks on the basis of QoS requirements into n groups (where, n represents total number of hosts). In this study we present a new scheduling heuristic based on the traditional Sufferage heuristic (Maheswaran *et al.*, 1999)

**Corresponding Author:** Ehsan Ullah Munir, School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

by dividing the tasks into two groups following the literature (Xiaoshan *et al*., 2003). Moreover, we perform the comparative analysis of current heuristics algorithms for independent task scheduling in gird computing based on earliest completion time of a set of jobs. The results are consistent with our hypothesis that tasks should be divided into two groups instead of n to save the computational resources of the Grid and maintaining a low overall makespan.

## QoS SUFFERAGE HEURISTIC

Here we will present a thorough discussion and example of proposed scheduling heuristic, where modification is employed in the Sufferage heuristic by introducing network bandwidth as QoS factor. QOS in an extensive concept, that varies from application to application. For example, QoS for CPU may mean its speed and processing power; similarly QoS of a network may mean its bandwidth and latency. Figure 1 summarizes some other identified quality of service parameters in computational Grid. In the current study we have considered network bandwidth as a decisive factor in our scheduling heuristic.

In QoS guided Min-min heuristic tasks are divided into two groups: High QoS and Low QoS. Min-min is applied on both groups. Tasks in high QoS group are given precedence in execution over tasks in low QoS group (Xiaoshan *et al*., 2003). The basic idea of QoS priority grouping algorithm presented in the literature (Dong *et al*., 2006) is that based on the number of hosts on which tasks can execute; they can be divided into n groups, where, n represents total number of hosts in Grid computing environment. And then task scheduling is performed for each group by applying Sufferage algorithm independently. However, for certain scenarios where tasks cannot be divided into n groups, it underperforms.

Our proposed heuristic divides the tasks into two groups according to high QoS and low QoS requirements, with the observation that tasks with high QoS requirements can only be executed on hosts with high QoS provision. For each task it calculates the earliest completion time and the machine that would obtain it. If machine is unassigned the task is assigned to that machine. Otherwise, if the machine is already busy the task is assigned to a machine with highest sufferage value after calculation. Here, sufferage value is equal to the second earliest completion time minus earliest completion time.

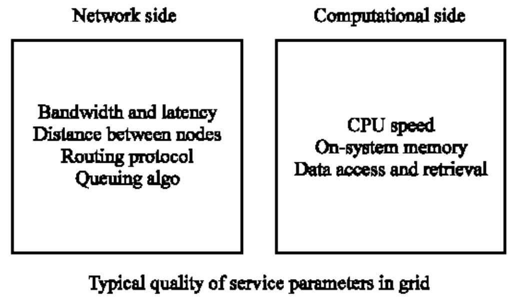


Fig. 1: QoS parameters distributed between network and computational sides in Grid

Throughout the study we use the terminology by Braun *et al*. (2001) and Maheswaran *et al*. (1999). The expected execution time $e_{ij}$ of task $t_i$ on machine $m_j$ is defined as the amount of time taken by $m_j$ to execute $t_i$ given $m_j$ has no load when $t_i$ is assigned. The expected completion time $c_{ij}$ of task $t_i$ on machine $m_j$ is defined as the wall-clock time at which $m_j$ completes $t_i$ (after having finished any previously assigned tasks). Let m be the total number of machines in the heterogeneous computing suite and K is the set containing the tasks that will be used in a given test set for evaluating heuristics in the study. Suppose the arrival time of the task $t_i$ be $a_i$ and the time $t_i$ begins execution be $b_i$. From the above definitions we have $c_{ij} = e_{ij}+b_j$. Let $c_i$ be the completion time for task $t_i$ and it is equal to $c_{ij}$ where, machine $m_j$ is assigned to execute task $t_i$. The makespan for the complete schedule is then defined as max $t_i \in K(c_i)$. The most important objective of Grid scheduling algorithm is to minimize the makespan. To proceed expected Execution Time to Compute (ETC) matrices are generated by following methods described by Braun *et al*. (2001) and Maheswaran *et al*. (1999). The pseudo-code for QoS Sufferage heuristic is given in Fig. 2.

A simple example is given below to illustrate the execution of QoS Sufferage heuristic and to make its comparison with QoS guided Min-min and QoS priority grouping. Task execution times of 9 tasks on 5 machines are recorded in Table 1 for simulation purpose, where each row represents the estimated execution times for a given task on corresponding machines. The machines are assumed to be idle for this case. In Table 1 X denotes that the machine doesn't have capability to perform that particular task due to its low QoS provision.

In this particular scenario, QoS Min-min heuristic gives a makespan of 20.7, QoS priority heuristic gives a makespan of 19.1 and QoS Sufferage heuristic gives a

(1) **for** all tasks $t_i$ in meta-task $M_v$ (in an arbitrary order)
(2) **for** all machines $m_j$ (in fixed arbitrary order)
(3) $c_{ij} = e_{ij} + r_j$ ($r_j$ = next available time of machine $m_j$)
(4) **do** until all tasks with high QoS request are mapped
(5) for each task $t_k$ with high QoS find a machine in the QoS qualified machine set that obtains the earliest completion time
(6) sufferage value = second earliest completion time-earliest completion time
(7) **if** machine $m_j$ is unassigned assign task $T_K$ to the machine, delete $t_k$ from $M_v$ and mark $m_j$ as busy
(8) **else**
(9) **if** sufferage value of task $t_i$ already assigned to machine $m_j$ is less than that of task $t_k$ unassign $t_i$ and assign $t_k$ to machine $m_j$ and send $t_i$ back to metatask
(10) delete task $t_k$ from $M_v$
(11) update $r_j$
(12) update $c_{ij}$ for all i
(13) **end do**
(14) **do** until all tasks with low QoS are mapped
(15) for each task $t_k$ in $M_v$ find a machine that obtains the earliest completion time
(16) sufferage value = second earliest completion time-earliest completion time
(17) **if** machine $m_j$ is unassigned assign task $t_k$ to the machine, delete $t_k$ from $M_v$ and mark $m_j$ as busy
(18) **else**
(19) **if** sufferage value of task $t_i$ already assigned to machine $m_j$ is less than that of task $t_k$ unassign $t_i$ and assign $t_k$ to machine $m_j$ and send $t_i$ back to metatask
(20) delete task $t_k$ from $M_v$
(21) update $r_j$
(22) update $c_{ij}$ for all i
(23) **end do**

Fig. 2: The QoS Sufferage heuristic

Table 1: ETC matrix where QoS Sufferage outperforms Qos Min-min and Qos Priority

| Tasks | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ |
|---|---|---|---|---|---|
| $t_1$ | X | X | X | X | 10.5 |
| $t_2$ | X | X | X | 7.5 | 11.0 |
| $t_3$ | X | X | X | 5.2 | 6.2 |
| $t_4$ | X | X | 14.7 | 8.0 | 12.8 |
| $t_5$ | X | X | 4.4 | 13.3 | 7.2 |
| $t_6$ | X | 3.8 | 5.9 | 5.8 | 3.4 |
| $t_7$ | X | 7.8 | 4.0 | 17.3 | 6.8 |
| $t_8$ | 15 | 9.0 | 5.6 | 7.1 | 6.4 |
| $t_9$ | 5.1 | 4.8 | 4.1 | 16.1 | 13.2 |

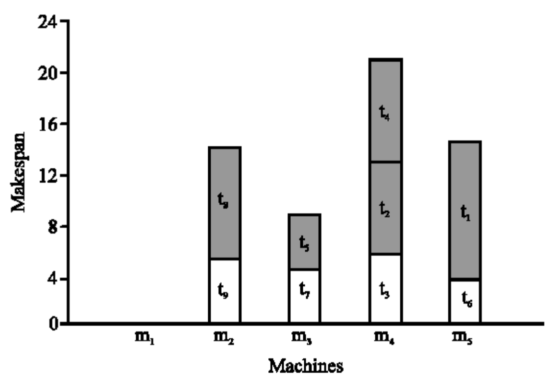


Fig. 3: Pictorial representation of QoS Min-min results

makespan of 16.7. Figure 3-5 give pictorial representations of the tasks assignments for QoS Min-min, QoS priority and QoS Sufferage, respectively.

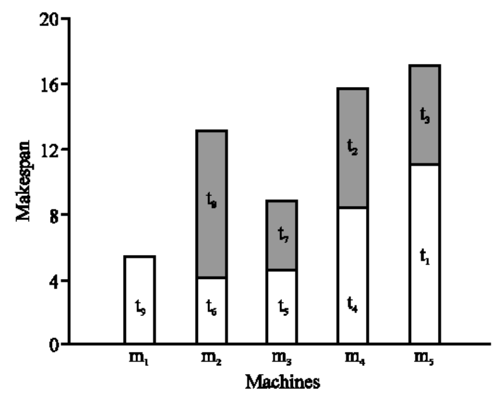


Fig. 4: Pictorial representation of QoS priority results

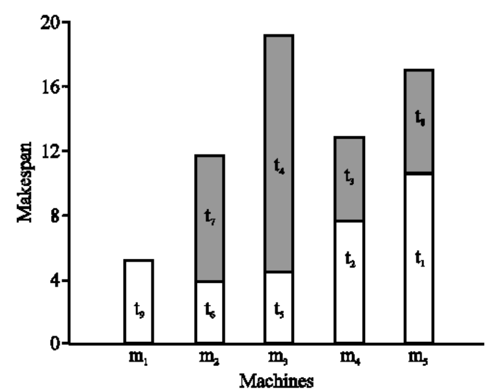


Fig. 5: Pictorial representation of QoS Sufferage results

From Fig 3-5 it is evident that the proposed QoS Sufferage heuristic outperforms the QoS Min-min and QoS priority heuristics based on makespan. Furthermore, it can be noted that the makespan given by the QoS Min-min is larger than the makespan obtained by the other two heuristics.

## SIMULATION AND RESULTS

A distributed heterogeneous computing system with user-defined number of machines and tasks is simulated to verify the results of QoS Sufferage heuristic and its comparison with the QoS guided Min-min and QoS priority heuristics. To carry out the simulation, we develop a simulator for task scheduling. The execution time of all the tasks taken into account namely, ETC matrix, is generated by using the same method presented by Braun *et al*. (2001) and Maheswaran *et al*. (1999). The rows of ETC matrix represent the execution times of each task on given machines. We are of the opinion that each scheduling heuristic can outperform other heuristics in a certain heterogeneous Grid environment and that each heuristic has certain limitations to perform its job in a certain scenario. The heuristic with shortest makespan is declared the best heuristic to perform task scheduling in Grid. The existing Sufferage algorithm is modified, which takes QoS factor into account to perform task scheduling. To verify the experimental results discussed in previous section ETC matrices with higher order are generated and tested for the proposed QoS Sufferage algorithm.

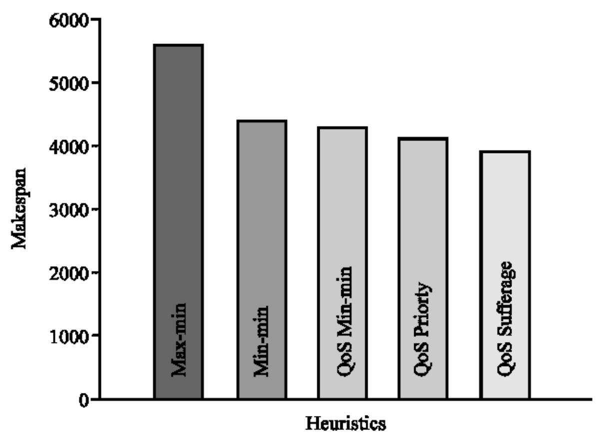


Fig. 6: Comparison of five heuristics in makespan. The QoS Sufferage gives a shortest makespan of 3860 for 512 tasks

The experimental results corresponding to ETC matrices of 100 tasks×10 machines, 512 tasks×16 machines and 1024 tasks×16 machines indicate that our proposed algorithm performs well and outperforms the QoS guided Min-min and QoS priority heuristics. However, the experimental results for 512 tasks×16 machines and 1024 tasks×16 machines only are being discussed here in detail, which are shown in Fig. 6-8. From these results it turns out that the QoS Sufferage performs very well and gives the shortest makespan of 3860 and 7460 for 512 and 1024 tasks, respectively. Verification of these results was carried out for many other scenarios with the same order of ETC matrices. Furthermore, it can be noted that the makespan of Max-min is much larger than all other heuristics considered for study. It is also suggested that the inclusion of more quality of service parameters can further reduce the overall makespan. We conclude by emphasizing that the QoS Sufferage is the best of all heuristics considered for this study in terms of shortest makespan.

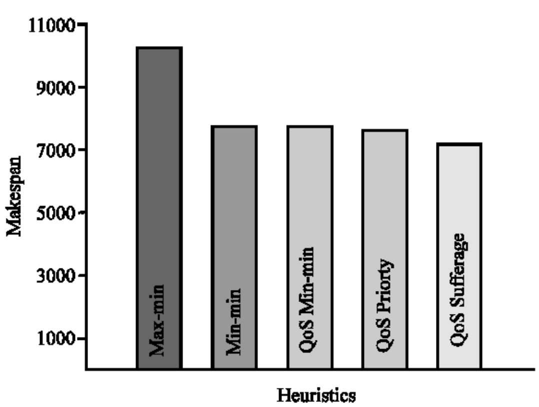


Fig. 7: Comparison of five heuristics in makespan. The QoS Sufferage gives a shortest makespan of 7460 for 1024 tasks

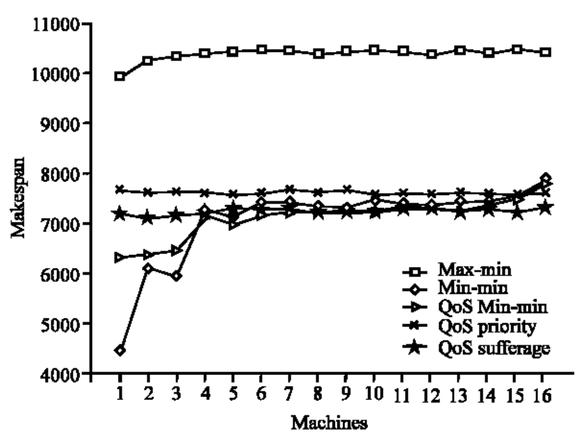


Fig. 8: Comparison in makespan with machine loads (Tasks = 1024)

## CONCLUSIONS

In this study, we have proposed a novel scheduling heuristic by considering QoS factor in scheduling and have proposed some modifications using existing Sufferage heuristic. We have compared our proposed scheme to other heuristics within a simulated Grid environment. Presented numerical simulation results confirm that proposed scheduling heuristic has a significant performance gain in terms of reduced makespan and outperforms all other heuristics considered here. We hope that this algorithm can lead to significant performance gain in variety of applications. In the present work we have considered only single-tier case for QoS (network bandwidth). However, in future, multi-tier QoS can be considered to maximize the performance of computational Grids.

## ACKNOWLEDGMENTS

This research is supported by the Key Program of the National Natural Science Foundation of China under Grant No. 60533110; the National Natural Science Foundation of China under Grant No. 60473075; the National Grand Fundamental Research 973 Program of China under Grant No. 2006CB303000; the Program for New Century Excellent Talents in University of China under Grant No. NCET-05-0333; the Heilongjiang Province Scientific and Technological Special Fund for Young Scholars under Grant No. QC06C033. Ph.D. scholarship for Mr. Ehsan Ullah Munir is provided by COMSATS Institute of Information Technology, Pakistan (CIIT).

## REFERENCES

Braun, T.D., H.J. Siegel and N. Beck, 1998. A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. IEEE Workshop on Advances in Parallel and Distributed Systems, West Lafayette, pp: 330-335.

Braun, T.D., H.J. Siegel and N. Beck, 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parall. Distrib. Comput., 61: 810-837.

Casanova, H., A. Legrand, D. Zagorodnov and F. Berman, 2000. Heuristics for scheduling parameter sweep applications in grid environments. In: Proceedings of the 9th Heterogeneous Computing Workshop, pp: 349-363.

Dong, F., J. Luo, L. Gao and L. Ge, 2006. A grid task scheduling algorithm based on QoS priority grouping. In: Proceedings of the 5th International Conference on Grid and Cooperative Computing, pp: 58-61.

El-Rewini, H., G. Theodore, Lewis and H.A. Hesham, 1994. Task Scheduling in Parallel and Distributed Systems. PTR Prentice Hall.

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

Foster, I. and C. Kesselman, 2001. The Anatomy of the grid: Enabling scalable virtual organizations. Int. J. High Perform. Comput. Applic., 5: 200-222.

Foster, I., 2002. What is the Grid? A three point checklist in grid today. Vol. 1. http://www.gridtoday.com/02/0722/100136.html.

Freund, R.F., M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J.D. Lima, M. Mirabile, L. Moore, B. Rust and H.J. Siegel, 1998. Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. In: 7th IEEE Heterogeneous Computing Workshop, pp: 184-199.

Ibarra, O.H. and C.E. Kim, 1977. Heuristic algorithms for scheduling independent tasks on non-identical processors. J. ACM., 24: 280-289.

Jinquan, Z., N. Lina and J. Changjun, 2005. A heuristic scheduling strategy for independent tasks on grid. In: Proceedings of the 8th International Conference on High-Performance Computing in Asia-Pacific Region.

Maheswaran, M., S. Ali, H.J. Siegel, D. Hensgen and R.F. Freund, 1999. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J. Parall. District Computed, 59: 107-131.

Sinnen, O., 2007. Task Scheduling for Parallel Systems. Wiley-Interscience.

Topucuoglu, H., S. Hariri and M.Y. Wu, 2000. Task scheduling algorithms for heterogeneous processors. In: Proceedings of the 8th Heterogeneous Computing Workshop, pp: 3-14.

Xiaoshan, H.E., X.H. Sun and G.V. Laszewski, 2003. QoS guided min-min heuristic for grid task scheduling. J. Comput. Sci. Technol. (Special Issue on Grid Computing), 18: 442-451.