Turkish Journal of Electrical Engineering and Computer Sciences

Volume 20 | Number 3 Article 11

1-1-2012

Adaptive QoS scheduling in a service-oriented grid environment

TAN FONG ANG

TECK CHAW LING

KEAT KEONG PHANG

TÜBİTAK

Turk J Elec Eng & Comp Sci, Vol.20, No.3, 2012, © TÜBİTAK
doi:10.3906/elk-0911-275

Adaptive QoS scheduling in a service-oriented grid environment

Tan Fong ANG*, Teck Chaw LING, Keat Keong PHANG
Department of Computer System and Technology, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur-MALAYSIA
e-mails: angtf@um.edu.my, tchaw@um.edu.my, kkphang@um.edu.my

Received: 15.01.2010

Abstract

The use of grid technology and web services for resource sharing has received tremendous attention in recent years. The merging of these 2 technologies is able to provide additional multiple types of services and functionalities. However, the problem of scheduling services to meet quality of service (QoS) requirements remains challenging. This paper proposes an adaptive QoS (AQoS) scheduling algorithm for service-oriented grid environments. AQoS uses benchmarking and curve-fitting based on historical records to estimate job length. Job length and users' QoS requirements are then used to make scheduling decisions. AQoS is able to maximize service availability, reliability, and resource utilization while minimizing total service execution time. Experimental results show that AQoS outperforms MIN-MIN and MAX-MIN algorithms by 10%-30% in terms of makespan and 5%-20% in terms of reliability.

Key Words: *Automatic deployment, dynamic scheduling, grid, service-oriented architecture, web services*

1. Introduction

In recent years, grid technology has proven to be a new paradigm for scientific applications that are resource-intensive. Coincidentally, web services have emerged as the next-generation framework for building distributed applications, largely due to the advances of wide area network (WAN) technologies. The growing number of web services and grid resources stimulates interest about the possibility of merging both of the technologies. Service-oriented architecture (SOA) is deemed to be one of the solutions for the architecture of integrating web services and grid resources. SOA is a set of principles that define an architecture that is loosely coupled and comprises service providers and service consumers, who interact according to a negotiated contract or interface [1]. SOA provides the standard that guarantees interoperability among the services in a grid environment. Moreover, it simplifies the integration of heterogeneous systems by using a loosely coupled model.

Although SOA has enabled the integration of web services and grid technology, it is still yet to mature, and substantial effort is required to enhance it. One of the challenges is the lack of quality of service (QoS)

*Corresponding author: Department of Computer System and Technology, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur-MALAYSIA

support in SOA [2]. QoS information, such as availability, reliability, accessibility, and cost, is decisive, since this information is needed for service selection and resource mapping. Therefore, it is necessary to include these QoS parameters to achieve better service and resource selection.

Scheduling is the main challenge in merging web services and grid technology. Presently, the scheduler only schedules a web services job based on a specified amount of resources. When these resources are fully utilized, job requests are added to the waiting queue. There is no mechanism to automatically deploy the required web services to other free resources to fill the request. This has explicitly reduced resource utilization and increased job makespans.

In conjunction, adaptive QoS (AQoS) for service-oriented grid environments has been proposed to minimize makespans and maximize job service reliability. AQoS uses benchmarking and curve-fitting based on historical records to estimate job length. Job length and QoS requirements are then used to make scheduling decisions. In this study, the AQoS algorithm is compared with other scheduling schemes, such as MIN-MIN and MAX-MIN, using different job arrival and job length distributions. Experimental results show that AQoS outperforms the MIN-MIN and MAX-MIN algorithms in terms of makespan and reliability for different job lengths and distributions.

The remainder of this paper is organized as follows. In Section 2, related works are discussed. In Section 3, the proposed algorithm is presented in depth. Section 4 details the system architecture. In Section 5, the experimental results are presented and analyzed. Section 6 concludes the paper.

2. Related works

Over the past decade, the grid has emerged as an active platform for large-scale applications in science and engineering. However, there are still many challenges for grid scheduling due to the dynamic nature and heterogeneity of the grid. In [3], reviews of the challenges of grid scheduling were identified. Heterogeneity, dynamic performance of resources, computation, and data separation challenges were identified as the main concerns of current research. In recent years, a number of grid scheduling algorithms have been proposed [4-12]. The goal of these algorithms is to improve the performance of the grid as well as to maximize the utilization of resources. Many heuristic approaches, such as MIN-MIN and MAX-MIN, have been proposed to match the job request to the resource.

The MIN-MIN scheduling algorithm groups all of the job requests in the queue into a set and then sorts the request with the shortest job length to be first in the queue. The job is then assigned to the resource that utilizes the minimum completion time. The process continues until all resources are assigned to the requests in the set. The MAX-MIN scheduling algorithm is similar to MIN-MIN, with the exception that the requests are sorted with the longest job length placed first in the queue. The job is then assigned to the resource that returns the minimum completion time [11].

Although the algorithms above are able to optimize the performance of the grid, most of these algorithms have not taken QoS attributes into consideration. Furthermore, the algorithms assume that the job size is known prior to submission. In addition, the algorithms only assign the resources based on the preinstalled services. Most of these algorithms are unable to deploy the jobs to new resources dynamically, since the new resources do not have services installed.

The problem of scheduling jobs to meet QoS requirements in service-oriented grid environments is challenging. Several studies have been conducted to address the issue of QoS in service-oriented grid environments [13-21]. In [13], a novel QoS guided task scheduling algorithm for grid computing was presented. The algorithm

is based on general adaptive scheduling heuristics that include QoS guidance. The algorithm considers the single dimension QoS attribute, which is the network, as the constraint to optimize the allocation of job resources. An algorithm called the hybrid particle swarm optimization algorithm was presented in [14] to resolve dynamic web services selection with QoS global optimal in grid workflow. The problem of dynamic web service selection with QoS global optimal was transformed into a multiobjective services composition optimization with QoS constraints. The best QoS service instance was selected from the participant services with different service qualities.

In [15,16], the authors developed a web service configuration net based on Petri nets and proposed an optimal algorithm to select the best configuration with the highest quality of service to meet users' nonfunctional requirements. The research was extended to formulate multiple attribute QoS optimization problems for a linear programming problem. Additionally, a generic SOA grid model that defines the application-specific QoS attributes was presented in [17]. A heuristic optimization scheduling algorithm was then proposed based on the defined aggregate of QoS attributes.

In terms of workflow systems, a web services-based QoS-aware workflow management system was developed in [18] to integrate with the existing grid technology. The system accepts the QoS requirements in Business Process Execution Language models and the QoS attributes are used in resource selection. In addition, [19] provided tools for adaptive service composition and provisioning, and [20] presented an adaptive resources provisioning scheme that maximizes network utilization and provider's revenues and minimizes request blocking probability while satisfying the required QoS.

In a service-oriented grid environment, web services are registered through a Universal Description, Discovery, and Integration (UDDI) registry. The discovery of web services is only based on the functional match. UDDI was not designed to represent and store the values of QoS attributes. In [21], the authors enhanced the functionalities of UDDI by storing QoS attributes in the UDDI. The QoS attributes were then used for better resource selection.

Once the job-to-resource mapping has been generated, the job can be deployed to the selected resources. The job deployment is another challenging issue that needs to be addressed. Various job deployment mechanisms have been explored in grid computing. In [22], the authors deployed web services over a distributed hash table and created replicas of a service on demand. Architecture that allows the deployment of services in a group of computers connected in a peer-to-peer fashion was proposed in [23].

More recently, Kecskemeti et al. [24] proposed an automatic service deployment using virtualization. This approach supports both the on-demand deployment and the self-healing services. Other similar works in deployment using virtualization were discussed in [25,26].

In relation to the above, this paper differs primarily from previous research in 4 ways. First, an adaptive mechanism is used for the service and resource selection. The previous works only selected the resources with the preinstalled service as the candidate. The proposed work selects all of the resources as the candidate. Second, a QoS registry is developed on top of the UDDI to store the QoS attributes. The QoS attributes will serve as a key index to differentiate the resources with identical web service functionality. Third, an AQoS scheduling algorithm for a service-oriented grid environment is proposed. Fourth, an automatic deployment mechanism is developed to allow dynamic service deployment. This mechanism automates the service deployment to the resources without the need for a web server.

3. Adaptive scheduling algorithm

In this section, the AQoS scheduling algorithm is presented. The grid, G, in the research consists of multiple sites, and each site, S_i, consists of a different number of computational resources. The computational resources can be divided into primary resources, p_j, and secondary resources, r_k.

$$G = \{S_1, S_2, \ldots, S_m\}, S_i, 1 \leq i \leq m, \tag{1}$$

$$S_i = \{p_1, p_2, \ldots, p_n\} U \{r_1, r_2, \ldots, r_q\}, \quad n, q \in \mathbb{N}, \tag{2}$$

where $\mathbb{N}$ is a natural number.

$$p_j \cap r_k = \emptyset, 1 \leq j \leq n, 1 \leq k \leq q, \tag{3}$$

where p_j is primary resources such as the server and cluster with a preinstalled web server and high-end central processing unit (CPU), and r_k is secondary resources such as a desktop computer without a web server installed and a low-end CPU. In each primary resource, there are a number of preinstalled web services, $w_{g,q}$.

$$p_g = \{w_{g,1}, w_{g,2}, \ldots, w_{g,q}\}, 1 \leq g \leq n, q \in \quad \mathbb{N} \tag{4}$$

$$w_{i,j} = w_{l,m}, \text{ when } i \neq l \tag{5}$$

$$\text{and } w_{i,j} \cap w_{l,m} = \emptyset, \text{ when } i = l \tag{6}$$

The same web service functionality is available in different primary resources and will not be duplicated in the same primary resource.

Figure 1 illustrates the AQoS scheduling algorithm. The first stage of the algorithm involves resource selection. First, web service requests, $rq_{t,i}$, are processed for each time interval; t is the interval number and $i$is the sequence number. Each request consists of web service functionality, parameter values, and QoS requirements. The QoS requirements include the minimum CPU speed, memory required, bandwidth required, and the deadline of the requests.

The time interval used in the research was an arbitrary value. The value selected for the time interval should be less than the execution time of a web service. In addition, for each time interval, the scheduler is able to retrieve the latest resource information from the QoS registry. Batch mode scheduling is used for the algorithm since the requests are processed at a predefined interval.

After that, for each request in time interval t, the corresponding estimated relative execution time, $re_{t,i}$, is computed. $re_{t,i}$ serves as the estimated job length for each request. The job length can be estimated using the combination of application benchmarking and resource benchmarking. Application benchmarking estimates the job length by running the same web service with different parameter values on the same resource. Curve-fitting is then used to construct a mathematical function that has the best fit for the historical records. The curve can be linear, exponential, or polynomial. The mathematical function is then used to estimate the job length of the same web service with different parameter values.

Input:
- $rq_{t,i}$, where $rq_{t,i}$ is service request in time interval t
- $p_j, r_k \in S_i$, where S_i is a set of resources in site i

Output:
- SC: A schedule of $rq_{t,i}$ into p_j or r_k

For each time interval t:
service requests in waiting state at time t
let $rq_{t,i}$ is the service request at time t
compute the estimated relative execution time, $re_{t,i}$

Sort the list $rq_{t,i}$ in descending order by $re_{t,i}$, and generate a sorted list, SL_t
Compute the mean, me_t

For each service request in SL_t:
select a list of available { p_j } and sort the list in descending order by CPU benchmark
select a list of available { r_k } and sort the list in descending order by CPU benchmark

if mean $re_{t,i} > me_t \times z$
assign $rq_{t,i}$ to p_j with the shortest completion time
otherwise
assign $rq_{t,i}$ to r_k with the shortest completion time

Figure 1. Adaptive QoS scheduling algorithm.

Resource benchmarking estimates the job length by running the same set of web services with the same parameter values on a different set of resources. The CPU speed is used as the metric to differentiate the resources since the web services used in the experiment are computing intensive. A ratio is then computed using the historical records, and the job length can be estimated when the web service is executed on resources with different CPU speeds. Once $re_{t,i}$ is computed for each request in interval t, the requests are sorted in descending order by comparing each $re_{t,i}$ value.

$$SL_t = \{re_{t,i}, re_{t,(i+1)}, ..., re_j\}, \text{ where } re_{t,i} \geq re_{t,j}, i < j \tag{7}$$

Following that, the mean of the estimated job length for each interval, me_t, is computed as below. x is the number of service requests at time interval t. me_t is used as the decision value to assign the requests to either primary resources or secondary resources.

$$me_t = (\sum_{i=0}^{x} re_{i,t})/x \tag{8}$$

Subsequently, for each service request in SL_t, a list of available primary resources that match the service request's QoS requirements is selected and the list is sorted in descending order by comparing the CPU benchmarks [27]. A similar approach is performed for the available secondary resources. The decision to assign the requests to

the primary or secondary resources is made by comparing the $re_{t,i}$ with the aggregate of multiplying me_t and z. If $re_{t,i}$ is greater than the aggregate, the request is assigned to the primary resource that can complete the request in the shortest time. Otherwise, the request is assigned to the secondary resource that can complete the request in the shortest time. The z value is computed using an empirical method with an increment or decrement of 0.1. The best z value will generate a schedule list with the shortest makespan.

$$z = z \pm 0.1, \text{ where } z = 1 \tag{9}$$

4. System architecture and testbed discussion

This section describes the system architecture and the testbed used. Figure 2 depicts the architecture of the proposed algorithm [28]. As shown in the Figure, the user first submits web service requests together with QoS requirements to the user interface (UI). The manager then retrieves the web service requests from the UI at each time interval t.

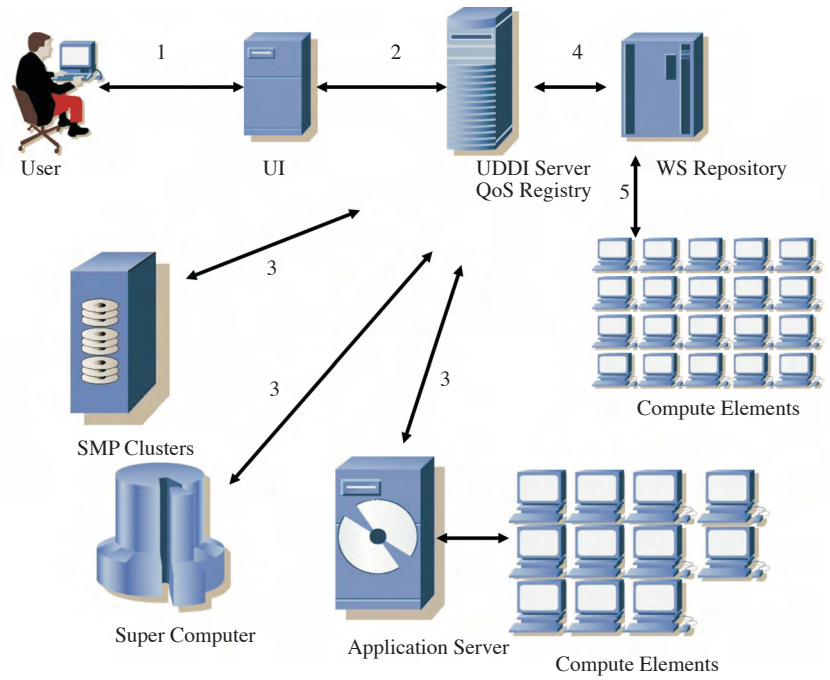

Figure 2. Automatic web services deployment.

After that, the manager searches the related web services from the UDDI server in terms of their functionalities. The manager also retrieves the QoS attributes of the resources from the QoS registry. The QoS registry stores the dynamic information about the current state of the resources on the grid. Periodically, an agent in each resource sends the latest resource information to the QoS registry.

Once the manager collects enough information, the scheduler generates a service request to the resource mapping list. The manager will then dispatch the service requests to the selected resources. Since the primary resources are preinstalled with the services, no deployment is required. The manager invokes an asynchronous communication to the web service end point because the execution of the services requires a significant amount

of time to process. The asynchronous communication invocation is used to allow other threads to continue their executions without being blocked.

The automatic services deployment mechanism is used to deploy the service to the secondary resource. Different approaches are used in deploying the services to different platforms. The first approach deploys the services to the .NET framework resources. The manager first retrieves the service *asmx* file from the web services repository and deploys it on the resource. The agent in the resource creates a web service end point with an IP address and port number that map to the *asmx* file. Http.sys architecture is used as the low-level HTTP protocol stacks for the service deployment. When http.sys receives a request, it can forward it directly to the correct work process. Http.sys is capable of caching responses directly within the kernel. This improves the overall throughput and performance.

The second approach is implemented in the Java 2 Platform Enterprise Edition (J2EE) environment. In this approach, the manager first retrieves the JAR file from the web services repository and deploys it on the resource. The JAX-WS 2.0 Endpoint Class is used to create the web service end point. The JAR file contains the web service descriptions and implementation. The web service binds to an end point with a WSDL address. Since the execution of the services requires a significant amount of processing time, the asynchronous communication invocation is again used to invoke the services. Once the job is completed, the output is stored in the UI server and the user will be notified.

The automatic web services deployment architecture was implemented in .NET and J2EE environments. Since the web services manage to solve the interoperability problem, the web services can be deployed easily to different platforms.

An experimental testbed was developed to execute the service testing in a service-oriented grid environment. The testbed allows the registration of different types of web services as well as new resources. Parametric types of web services were selected for the experiment, in which the execution times of these services were affected by the parameter values. In addition, most of these web services consist of Monte Carlo implementations. The Monte Carlo method generates random sampling and is very computing-intensive when the sample size is huge. Monte Carlo is normally used for benchmarking.

5. Experimental results

To evaluate the performance of AQoS, 20 primary resources, which consisted of dual core and quad core servers and clusters, were configured. The range of the CPU speed was 2.0-2.66 GHz. In addition, 20 secondary resources, which consisted of dual core desktop computers, were configured. The CPU speed was around 1.86 GHz. These resources included those computers in the laboratories that were underutilized. All resources were located in 3 different administrative domains and running in a .NET or J2EE environment.

Several experiments were conducted using MIN-MIN, MAX-MIN, and AQoS based on different test cases: 1) random job arrival with random job length, 2) random job arrival with large gap job length, 3) random job arrival with job length in Poisson distribution, 4) Poisson arrival with random job length, 5) Poisson arrival with large gap job length, and 6) Poisson arrival with job length in Poisson distribution. The time interval used for the experiment was 20 s per interval. The size of the job length was based on the parameter values. For the AQoS scheduling algorithm, the experiments were conducted using different fixed values of z for the same test case, plus an experiment with dynamic z values where the value of z was changed at each interval during the run time.

Table 1 shows the results of the makespan and the percentage of improvement for different experimental

Table 1. Makespan and percentage of improvement for different test cases.

Test	Job description	Makespan (s)							% improvement of best AQoS as compared to MIN-MIN	% improvement of best AQoS as compared to MAX-MIN
		MIN-MIN	MAX-MIN	AQoS (z = 1)	AQoS (z = 0.9)	AQoS (z = 0.6)	AQoS (z = 0.5)	AQoS (dynamic z)		
1	Random job length (random job arrival)	8567.14	8486.33	8076.84	**7320.54**	7798.14	7906.33	7462.56	14.6%	13.7%
Test	Job description	MIN-MIN	MAX-MIN	AQoS (z = 2.1)	AQoS (z = 2)	AQoS (z = 1.4)	AQoS (z = 1.1)	AQoS (dynamic z)	% improvement of best AQoS as compared to MIN-MIN	% improvement of best AQoS as compared to MAX-MIN
2	Large gap job length (random job arrival)	54,078.14	50,427.11	50,921.94	42,257.61	44,801.28	48,386.04	**38,568.09**	28.7%	23.5%
Test	Job description	MIN-MIN	MAX-MIN	AQoS (z = 1)	AQoS (z = 0.9)	AQoS (z = 0.7)	AQoS (z = 0.5)	AQoS (dynamic z)	% improvement of best AQoS as compared to MIN-MIN	% improvement of best AQoS as compared to MAX-MIN
3	Poisson distribution job length, λ = 40,000,000 (random job arrival)	839.91	781.56	1279.56	**697.88**	752.39	766.97	718.62	16.9%	10.7%
Test	Job description	MIN-MIN	MAX-MIN	AQoS (z = 1.2)	AQoS (z = 1)	AQoS (z = 0.5)	AQoS (z = 0.4)	AQoS (dynamic z)	% improvement of best AQoS as compared to MIN-MIN	% improvement of best AQoS as compared to MAX-MIN
4	Random job length (Poisson arrival λ = 7)	35,071.66	34,077.42	30,899.84	**30,615.75**	30,675.76	32,942.19	**30,615.75**	12.7%	10.2%
Test	Job description	MIN-MIN	MAX-MIN	AQoS (z = 1.7)	AQoS (z = 1.5)	AQoS (z = 1)	AQoS (z = 0.9)	AQoS (dynamic z)	% improvement of best AQoS as compared to MIN-MIN	% improvement of best AQoS as compared to MAX-MIN
5	Large gap job length (Poisson arrival λ = 11)	88,922.1	84,534.83	83,790.59	75,612.52	79,624.42	81,617.48	**72,684.17**	18.3%	14.0%
Test	Job description	MIN-MIN	MAX-MIN	AQoS (z = 1)	AQoS (z = 0.9)	AQoS (z = 0.8)	AQoS (z = 0.7)	AQoS (dynamic z)	% improvement of best AQoS as compared to MIN-MIN	% improvement of best AQoS as compared to MAX-MIN
6	Poisson distribution job length, λ = 100,000,000 (Poisson arrival λ = 9)	3116.51	3040.82	5750.19	4070.43	2680.29	2855.32	**2589.34**	17.0%	14.8%

cases using the 3 different algorithms. The makespan was measured in seconds. The results show that AQoS outperformed the MIN-MIN and MAX-MIN algorithms by 10%-30% in terms of makespan. Table 1 also shows that the dynamic z value outperformed MIN-MIN and MAX-MIN by a significant margin. These results are shown in Figures 2-8.

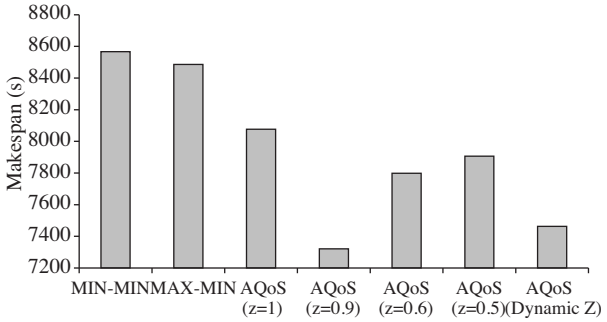

Figure 3. Random job arrival with random job length.

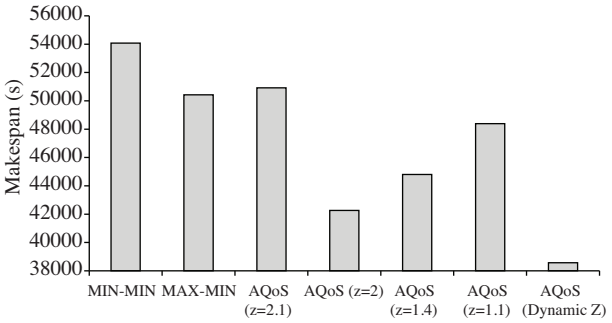

Figure 4. Random job arrival with large gap job length.

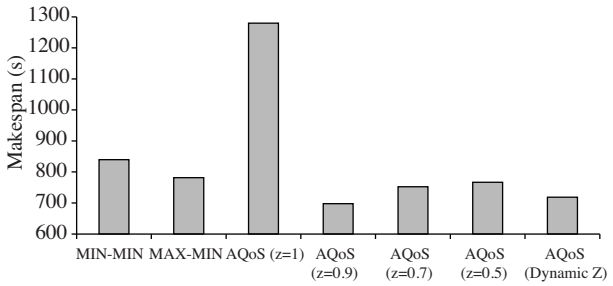

Figure 5. Random job arrival with job length in Poisson distribution.

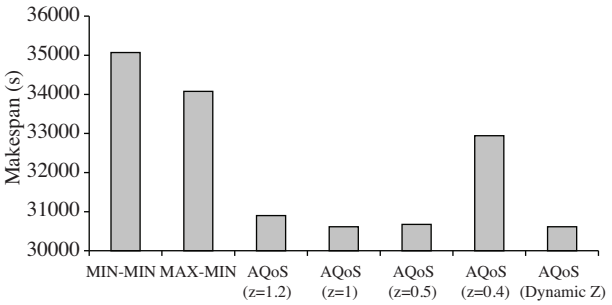

Figure 6. Poisson arrival with random job length.

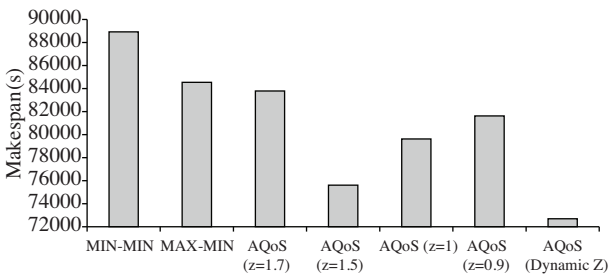

Figure 7. Poisson arrival with large gap job length.

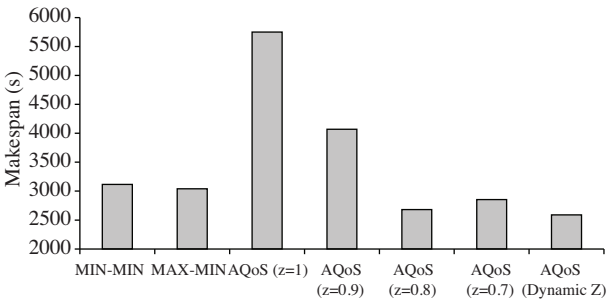

Figure 8. Poisson arrival with job length in Poisson distribution.

In addition to the measurement of makespan, an experiment to compare the reliability of the 3 different algorithms was conducted. The deadline value was added as the QoS requirement to each service request. Table 2 shows the results for reliability for different job lengths with Poisson arrival and the percentage of improvement. These results show that AQoS outperformed the MIN-MIN and MAX-MIN algorithms by 5%-20% in terms of reliability. The results are also shown in Figure 9 in the Appendix.

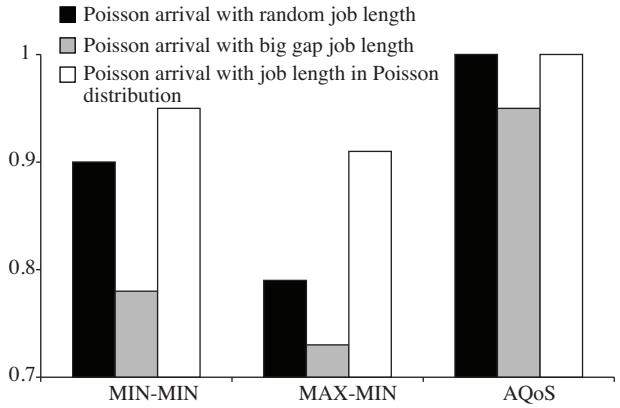

Figure 9. Service reliability for different job lengths with Poisson arrival.

Table 2. Reliability and percentage of improvement for different test cases.

Test	Job description	MIN-MIN (reliability)	MAX-MIN (reliability)	AQoS (reliability)	% improvement of AQoS as compared to MIN-MIN	% improvement of AQoS as compared to MAX-MIN
1	Random job length (Poisson arrival $\lambda = 7$) with deadline (estimation)	0.9	0.79	1	10.0%	21.0%
2	Large gap job length (Poisson arrival $\lambda = 11$) with deadline (estimation)	0.78	0.73	0.95	12.9%	18.2%
3	Poisson distribution job length, λ=100,000,000 (Poisson arrival $\lambda =$ 6) with deadline (estimation)	0.95	0.91	1	5.0%	9.0%

6. Conclusion

In this paper, an AQoS scheduling algorithm for service-oriented grid environments was presented. The major contributions of the paper include the usage of an adaptive mechanism for the service and resource selection, development of a QoS registry on top of the UDDI to store the QoS attributes, proposal of an AQoS scheduling algorithm for service-oriented grid environments, and presentation of an automatic deployment mechanism to dynamically deploy web services to secondary resources. Experimental results showed that AQoS minimized the makespan by 10%-30% and maximized the reliability by 5%-20% as compared to MIN-MIN and MAX-MIN. For future works, the proposed AQoS is currently being extended to include rescheduling and resource reservation.

Acknowledgment

This work was partly financially supported by the Malaysia eScience Fund (11-02-03-1041).

References

[1] M. Mansukhani, Service Oriented Architecture White Paper, Palo Alto, CA, USA, Hewlett-Packard, 2005.

[2] Y. Tang, Y. Yang, M. Zhao, L. Yao, Y. Li, "CoS-based QoS management framework for grid services", Sixth International Conference on Grid and Cooperative Computing, pp. 451-455, 2007.

[3] F. Dong, S.G. Akl, Scheduling Algorithms for Grid Computing: State of the Art and Open Problems, Technical Report 2006-504, School of Computing, Queen's University, Ontario, Canada, 2006.

[4] Y.C. Lee, A.Y. Zomaya, "Practical scheduling of bag-of-tasks applications on grids with dynamic resilience", IEEE Transactions on Computers, Vol. 56, pp. 815-825, 2007.

[5] N.D. Doulamis, A.D. Doulamis, E.A. Varvarigos, T.A. Varvarigou, "Fair scheduling algorithms in grids", IEEE Transactions on Parallel and Distributed Systems, Vol. 18, pp. 1630-1648, 2007.

[6] K. Etminani, M. Naghibzadeh, "A min-min max-min selective algorithm for grid task scheduling", IEEE/IFIP International Conference in Central Asia on Internet, pp. 1-7, 2007.

[7] C. Du, X.H. Sun, M. Wu, "Dynamic scheduling with process migration", International Symposium on Cluster Computing and the Grid, pp. 92-99, 2007.

[8] T.F. Ang, W.K. Ng, T.C. Ling, L.Y. Por, C.S. Liew, "A bandwidth-aware job grouping-based scheduling on grid environment", Information Technology Journal, Vol. 8, pp. 372-377, 2009.

[9] J.K. Kim, S. Shivle, H.J. Siegel, A.A. Maciejewski, T.D. Braun, M. Schneider, S. Tideman, R. Chitta, R.B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, S.S. Yellampalli, "Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment", Journal of Parallel and Distributed Computing, Vol. 67, pp. 154-169, 2007.

[10] N. Muthuvelu, I. Chai, C. Eswaran, "An adaptive and parameterized job grouping algorithm for scheduling grid jobs", 10th International Conference on Advanced Communication Technology, pp. 975-980, 2008.

[11] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", Journal of Parallel and Distributed Computing, Vol. 61, pp. 810-837, 2001.

[12] E. Caron, V. Garonne, A. Tsaregorodtsev, "Definition, modelling and simulation of a grid computing scheduling system for high throughput computing", Future Generation Computer Systems, Vol. 23, pp. 968-976, 2007.

[13] X.S. He, X.H. Sun, G. von Laszewski, "QoS guided min-min heuristic for grid task scheduling", Journal of Computer Science and Technology, Vol. 18, pp. 442-451, 2003.

[14] C. Hu, M. Wu, G. Liu, W. Xie, "QoS scheduling algorithm based on hybrid particle swarm optimization strategy for grid workflow", Sixth International Conference on Grid and Cooperative Computing, pp. 330-337, 2007.

[15] P.C. Xiong, Y.S. Fan, M.C. Zhou, "QoS-aware web service configuration", IEEE Transactions on Systems, Man, and Cybernetics, Vol. 38, pp. 888-895, 2008.

[16] P.C. Xiong, Y.S. Fan, M.C. Zhou, "Web service configuration under multiple quality-of-service attributes", IEEE Transactions on Automation Science and Engineering, Vol. 6, pp. 311-321, 2009.

[17] M. Wahib, A. Munawar, M. Munetomo, A. Kiyoshi, "SOAG: Service oriented architectured grids and adoption of application specific QoS attributes", IEEE Grid Computing Conference, pp. 346-351, 2008.

[18] L. Guo, A.S. McGough, A. Akram, D. Colling, J. Martyniak, M. Krznaric, "Enabling QoS for service-oriented workflow on GRID", International Conference on Computer and Information Technology, pp. 1077-1082, 2007.

[19] Q.Z. Sheng, B. Benatallah, Z. Maamar, A.H.H. Ngu, "Configurable composition and adaptive provisioning of web services", IEEE Transactions on Service Computing, Vol. 2, pp. 34-49, 2009.

[20] A. Filali, A.S. Hafid, M. Gendreau, "Adaptive resources provisioning for grid applications and services", IEEE International Conference on Communications, pp. 186-191, 2008.

[21] C.C. Lo, D.Y. Cheng, P.C. Lin, K.M. Chao, "A study on representation of QoS in UDDI for web services composition", International Conference on Complex, Intelligent and Software Intensive Systems, pp. 423-428, 2008.

[22] C.P. Gavaldà, P.G. López, R.M. Andreu, "Deploying wide-area applications is a snap", IEEE Internet Computing, Vol. 11, pp. 72-79, 2007.

[23] D. Lazaro, J.M. Marques, J. Jorba, "An architecture for decentralized service deployment", International Conference on Complex, Intelligent and Software Intensive Systems, pp. 327-332, 2008.

[24] G. Kecskemeti, P. Kacsuk, G. Terstyanszky, T. Kiss, T. Delaitre, "Automatic service deployment using virtualisation", 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 628-635, 2008.

[25] B. House, P. Marshall, M. Oberg, H.M. Tufo, "Grid service hosting on virtual clusters", Proceedings of the 9th IEEE/ACM International Conference on Grid Computing, pp. 304-309, 2008.

[26] L. Wang, G. von Laszewski, J. Tao, M. Kunze, "Grid virtualization engine: design, implementation, and evaluation", IEEE Systems Journal, Vol. 3, pp. 477-488, 2009.

[27] PassMark, PassMark Software 2009, Sydney, Australia, PassMark® Software Pty Ltd. Available at http://www.cpubenchmark.net/.

[28] A.T. Fong, L.T. Chaw, P.K. Keong, P.L. Yee, "Automatic web services deployment", World Congress on Computer Science and Information Engineering, pp. 315-319, 2009.