Efficient Resource Selection Algorithm for Enterprise Grid Systems.

Albert Zomaya

University of Sydney

585 PUBLICATIONS **4,072** CITATIONS

SEE PROFILE

Jemal Abawajy

Deakin University

250 PUBLICATIONS **1,306** CITATIONS

SEE PROFILE

Efficient Resource Selection Algorithm for Enterprise Grid Systems

Wan Nor Shuhadah Wan Nik, Bing Bing Zhou, Albert Y. Zomaya
School of Information Technologies
University of Sydney
Sydney, Australia
{wwan3494,bing.zhou,albert.zomaya}@sydney.edu.au

Jemal H. Abawajy
School of Information Technology
Deakin University
Melbourne, Australia
jemal.abawajy@deakin.edu.au

Abstract—**This paper addresses a resource selection problem for applications that update data in enterprise grid systems. The problem is insufficiently addressed as most of the existing resource selection approaches in grid environments primarily deal with read-only job. We propose a simple yet efficient algorithm that deals with the complexity of resource selection problem in enterprise grid systems. The problem is formulated as a Multi Criteria Decision Making (MCDM) problem. Our proposed algorithm hides the complexity of resource selection process without neglecting important components that affect job response time. The difficulty on estimating job response time is captured by representing them in terms of different QoS criteria levels at each resource. Our experiments show that the proposed algorithm achieves very good results with good system performance as compared to existing algorithms.**

Keywords-resource selection; enterprise grid computing; asynchronous replication; QoS; performance prediction

I. INTRODUCTION

In an enterprise grid computing environments, users have access to multiple resources that may be distributed geographically [1]. Also, in order to increase the availability and the performance of enterprise grid computing systems, data may be replicated across enterprise grid resources. In such environments, enterprise grid applications that update data [2] may introduce conflict. Further, when this type of application is deployed in a very large corporation, it may involve a significant number of resources [3]. The value of the execution result must be propagated to other resources to maintain data consistency. In case of conflicting jobs, resource is only allowed to execute the jobs when it holds or gets the propagation of the latest value of data to avoid conflicts. However, the sequence and arrival time of job propagation is unpredictable due to the various factors of heterogeneous systems such as different latencies on different network links, different processing speeds of the resources, etc. This scenario makes the prediction of job response time very difficult. However, this difficulty can be reduced if one can ensure the correctness of job execution despite of the unpredictable sequence and arrival time of job propagation among resources before the prediction of job response time is made. Thus, appropriate resource selection for grid applications is a fundamental issue in achieving high performance on enterprise grid computing with data replications.

In this paper, we propose an automatic resource selection approach for executing enterprise grid applications that update data. The proposed approach is based on a new resource performance prediction technique. The technique depends on resource performance with respect to job response time. The resource performance is captured in terms of several *QoS* criteria that affect job response time in asynchronous replicated system. Experimental analysis shows that the proposed approach selects the best available resources for executing jobs.

In summary, we highlight three main contributions of this paper. First, we provide a new prediction technique on resource performance with respect to job response time by addressing it in terms of several *QoS* criteria. Second, we defined each of these criterions so that they will reflect important factor that affect job response time in asynchronous replicated system. Finally, we solved the problem on selecting the best resource which can process job in shortest time by evaluating the trade-offs between these criteria at each resources. This is done by exploiting the TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) and entropy method.

The rest of the paper is structured as follows: Related work and background are presented in Section 2 and Section 3, respectively. Then, the proposed grid resource selection algorithm is discussed in Section 4 while experimental configuration is presented in Section 5. Section 6 presents the results and discussions. Finally, Section 7 concludes.

II. RELATED WORK

Generally, there are two types of replication schemes in distributed systems; *synchronous* and *asynchronous*. In this paper, we focus on the asynchronous replication which is desirable for data with weaker consistency requirement to achieve better performance. This technique allows job to commit at one replica and later propagates the update value to other replica after the result of job execution has been returned to the user.

A resource selection model based on Decision Theory is proposed in [4] and [5]. These works are similar to our work in the sense that they apply a decision theory to select the

978-0-7695-4428-1/11 $26.00 © 2011 Crown Copyright
DOI 10.1109/ISPA.2011.25

best resource in grid systems. However, none of them provide a detailed approach on how to predict the job response time for update-intensive applications. Existing research primarily focuses on resource selection by read-only jobs [6], [7], [8].

In [8], the proposed resource selection mechanism is based on two main algorithms, namely Minimum Execution Time (MET) and Minimum Completion Time (MCT) heuristics. In these algorithms, the processing time for all jobs that are queued at resources is expressed as *resource availability* which is defined as the earliest time that the resource can complete the execution of all jobs that has previously assigned to it. In MET, each job is assigned to a resource based solely on resource processing speed and job size without considering *resource availability*. In contrast, MCT assigns each job to a resource that has minimum completion time for that job by taking into account the *resource availability*. Indeed, there is little work on mechanisms for best resource selection especially in enterprise grid systems and our work is the first attempt that provides a mechanism on how to select the best resources in the presence of job that update data.

In this paper, the *Update Ordering* (UO) approach [9] will be adopted to deal with the unpredictable job sequence and arrival time. This approach implements the *Unique Sequence Number* (USN) which allows jobs to be executed in different, yet a sensible order at different resources while still ensures the correctness of replicated data. We also exploit the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) method [10].

III. Background

A. System Architecture

Fig. 1 shows the high-level enterprise grid architecture of interest. Users at Tier-1 send a job request to the User Broker (UB) at Tier-2 which then works together with Transaction Service (TS) at Tier-2 to select and allocate a job to an appropriate resource for execution at Tier-3.

The system consists of a set of data $D_{sys} = \{d_1, d_2, ..., d_m\}$ where m represents the total number of data in the system. For any particular data $d_k \in D_{sys}$, a set of its replica is denoted as

$$\left|\Re^k\right| = \left|\{s_1^k, s_2^k, ..., s_N^k\}\right|$$

That is, we assume that one resource holds one replica for any particular data d_k .

When a job updates a value of data replica at any particular resource in Tier-3, this job and its updated value must be propagated to other resources to maintain data consistency. When jobs are propagated to a group of resources by different resources concurrently, their arrival orders may be different. This results from various factors of heterogeneous systems such as different processing speeds of the machines on which the replicas reside in Tier-3, different latencies on different network links and so forth.

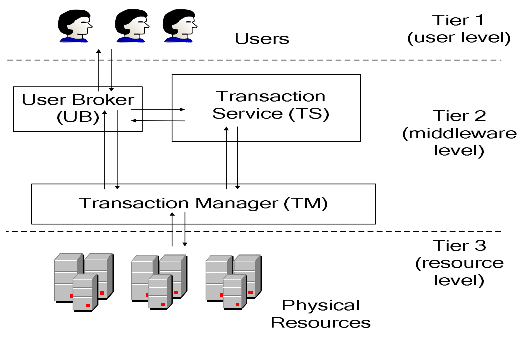


Figure 1. A 3-tier enterprise grid systems architecture.

Assuming an N-available resource in the system, $\Re^N = \{R_1, ..., R_N\}$, a job received by resource R_x directly from user agent/broker is said to be *originated* from R_x . When a request is *received* by a resource, its ordering constraint (i.e., *total, causal* and *commutative) is checked*. Once it's ordering constraint is satisfied, the job is ready to be *executed* by the resources. To deal with the unpredictable sequence and arrival time of job propagation, a *Unique Sequence Number* (*USN*) [9] is used to ensure the correctness of job execution at resources. For each data d_k , the value of *USN* is attached to a job and is propagated among resources during job propagation process. To be able to decide if the jobs can be executed, each resource keeps a variable in its local space to record the maximum USN executed so far for each data replica it holds. If a job arrived at resource holds the next USN, then this job is ready to be *executed*. Otherwise, the job is *deferred* until lower USN jobs are performed. In our framework, the functionality of the sequencer is performed by the collaboration of UB and TS. Also, we define:

$$\theta_{sys} = \{\{J^{d_1}\}, \{J^{d_2}\}, ... \{J^{d_m}\}\}$$

where

- $\{J^{d_k}\}$ is a set of job operation (i.e., *total, causal* and *commutative* [9]) for d_k with k=1, 2,…, m.
- $J^{d_k} = \{u_1^{d_k}, u_2^{d_k}, ..., u_n^{d_k}\}$, n representing the total number of job operation on d_k such that $u_1^{d_k}$ and $u_2^{d_k}$ are job operation on the same data d_k .

For example, if job u^{d_k} is a commutative with every job in J^{d_k} , then u^{d_k} is a commutative job. A commutative job implies that the order of its execution does not affect the state of replicas. If u^{d_k} conflicts with one job in J^{d_k} , u^{d_k} is a total job. When a job is a total job, it needs to be executed sequentially at all its replicas to ensure data correctness. Meanwhile, the causal operation job is based on the *happened-before* semantics which captures the potential *cause-effect* relation between two job events. Therefore, for any particular data d_k, the following definitions imply:

Definition 1: (total job operation set – $Total_{op}^{d_k}$). $Total_{op}^{d_k}$ contains all total job operation out of J^{d_k}.

Definition 2: (commutative job operation set – $Comm_{op}^{d_k}$). $Comm_{op}^{d_k}$ contains all commutative job operations out of J^{d_k}.

Definition 3: (causal job operation set – $Causal_{op}^{d_k}$). $Causal_{op}^{d_k}$ contains all caused-by operations out of J^{d_k}.

However, for two jobs $u_i^{d_k} \in Total_{op}^{d_k}$ and $u_i^{d_l} \in Total_{op}^{d_l}$, $u_i^{d_k}$ and $u_i^{d_l}$ are not conflicting to each other although they are conflicting in their own dataset since each of them is working on different data. With $Total_{op}^{sys}$, $Causal_{op}^{sys}$ and $Comm_{op}^{sys}$ are the total number of *total operation*, *causal operation* and *commutative operation* in the system respectively, therefore we have

$Total_{op}^{sys} = \sum_{i=1}^{N} Total_{op}^{d_i}$,

$Causal_{op}^{sys} = \sum_{i=1}^{N} Causal_{op}^{d_i}$ and $Comm_{op}^{sys} = \sum_{i=1}^{N} Comm_{op}^{d_i}$

with

$Total_{op}^{d_k} \cap Total_{op}^{d_l} = \phi$ and $Causal_{op}^{d_k} \cap Causal_{op}^{d_l} = \phi$

but

$Comm_{op}^{d_k} \cap Comm_{op}^{d_l} = Comm_{op}^{sys}$.

B. Use Case Example

Consider a business application which may run a Web shop application on enterprise grid platform. Its datasets may be replicated across distributed and heterogeneous resources. Suppose that the business customers along with the business staff are accessing and updating this dataset. The possible set of job operation on this dataset is denoted by G_{sys} = {*addProduct, deleteProduct, sendComment, replyComment, editPrice, addStock, editSupplier*}. Business customer and its staff are application users' who may access and update data at any particular time. *addProduct, deleteProduct* and *editPrice* are job operations on "Product ID" and "Price" data and are in conflict with each other. These jobs and their data may involve customer profiles and credit card information which must be handled carefully. For example, these jobs may be sent from business staff and at the same time their data are accessed by customers via web application during online shopping. Any conflict may result in an incorrect price that needs to be paid by web shop customers. Concurrent operations on this data are in conflict with each other and thus job operation *addProduct* and *deleteProduct* is categorized as $Total_{op}$ to ensure its correctness.

Meanwhile, *sendComment* and *replyComment* job operation on "Product Description" data is categorized as $Comm_{op}$ and $Causal_{op}$ respectively. That is, any comment on "Product Description" made by customer must be addressed accordingly by business staff's to meet the customer satisfaction. Further, any job operation on "Stock in Hand" and "Supplier" data (i.e. *addStock,* and *editSupplier*) is categorized as $Comm_{op}$ since these job operations do not conflict. User can tolerate some inconsistencies on these data. For example, user can still put an order for any particular product as long as the product is in stock or the supplier has changed.
In this example, different type of jobs that update data may present at large number of resources which may geographically distributed. In such scenario, resource selection process becomes a major problem that need to be addressed in order to execute job in low response time.

IV. Grid Resource Selection Algorithm

Fig. 2 shows the proposed resource selection algorithm. The algorithm first determines different ***QoS*** (i.e., freshness, efficiency and reliability) values for each resource. It then constructs a decision matrix *D* (step 4) and assigns weights to freshness, efficiency and reliability based on entropy (step 5).

A. Evaluating QoS Criterions

The freshness value is concerned with the execution of jobs in $Total_{op}^{d_k}$ and $Causal_{op}^{d_k}$ since these jobs may conflict with at least one job in the system. The freshness value is computed as follows:

$$freshness_{R_x} = \sum_{i=\beta}^{n} (size(u_i^{d_k}) / nl_{R_x,R_y}) \qquad (1)$$

where $n = \alpha - \beta$ is a total number of jobs queued at R_x which are in conflicts to each other with α and β are the job ID (i.e. USN) that last *originated* in the system and last *executed* at resource R_x for any particular data d_k respectively. $u_i^{d_k} \in Total_{op}^{d_k} \cup Causal_{op}^{d_k}$ is a job for a particular data d_k, nl_{R_x,R_y} is a network bandwidth between resource R_x and R_y with $R_x \neq R_y$ and R_y is an *originated* resource for $u_i^{d_k}$.

The *freshness levels* of replicas at resources can be estimated by how many job arrivals that the resource has missed or the time the resource needs to wait before it can start to process a new job assigned to it to avoid conflicts. In this case, resource R_x is said as the *freshest* resource in the system if $\alpha = \beta$. This definition is possible due to the fact that the implementation of USN is sufficient to guarantee the correctness (i.e. sequence) of job execution for

$Total_{op}$ and $Causal_{op}$ at each resource in asynchronous replicated system [9].

The efficiency value is defined as follows:

$$efficiency_{R_x} = \sum_{i=1}^{n} (size(u_i)/cs_{R_x}) \quad (2)$$

where n and cs_{R_x} are the number of waiting jobs and computing speed of resource R_x respectively. $size(u_i)$ is a size of job $u_i \in \theta_{sys}$.

Meanwhile, the reliability value is defined as follows:

$$reliability_{R_x} = (\frac{ActRT - EstRT}{ActRT})_{R_x} \quad (3)$$

where $ActRT$ and $EstRT$ are an average of actual and estimated response time at resource R_x respectively with $ActRT \geq EstRT$, $ActRT \neq 0$ and $EstRT \neq 0$. This criterion is defined due to the fact that it is most likely that the actual job response time is longer than its original estimates due primarily to delays. In real applications, delays may be caused by complex interactions among heterogeneous resource hardware, software, networking and security [11]. For each criterion ($freshness_{R_x}$, $efficiency_{R_x}$, $reliability_{R_x}$), lower value is more desirable as it can contribute to shorter job response time.

```
Algorithm ResourceSelection
BEGIN
1.  FOR each resource $R_i \in \Re^N$ DO
2.      Evaluate $freshness_{R_i}$, $efficiency_{R_i}$ and $reliability_{R_i}$
3.  ENDFOR
4.  Construct a decision matrix D as shown in (4).
5.  WeightAssignment ($freshness_{R_i}$, $efficiency_{R_i}$, $reliability_{R_i}$)
6.  Normalize D & its weight whose elements are defined by
        $z_{ij} = f_{ij} / \sqrt{\sum_{i=1}^{m} f_{ij}^2}$, $i=1,\ldots,m; j=1,\ldots,n.$
7.  Formulate the weighted normalized decision matrix whose elements are $x_{ij} = w_j z_{ij}$, $i=1,\ldots,m; j=1,\ldots,n.$
8.  Determine the positive ($a^+$) and the negative ($a^-$) ideal solutions as follows:
        $a^+ = \{(\min_i x_{ij} \mid j \in J) \mid i=1,\ldots,m\} = \{x_1^+, x_2^+, \ldots, x_n^+\}$
        $a^- = \{(\max_i x_{ij} \mid j \in J) \mid i=1,\ldots,m\} = \{x_1^-, x_2^-, \ldots, x_n^-\}$
9.  Calculate the separation measures for ideal and negative ideal solutions of each resources as follows:
        $R_i^+ = \sqrt{\sum_{j=1}^{n} (x_{ij} - x_j^+)^2}$, i = 1,…, m
        $R_i^- = \sqrt{\sum_{j=1}^{n} (x_{ij} - x_j^-)^2}$, i = 1,…, m
10. Calculate relative closeness of each resource to the ideal point as follows:
        $cl_i^+ = R_i^- / (R_i^- + R_i^+)$, $0 \leq cl_i^+ \leq 1$ and $i=1,\ldots,m.$
11. Rank the resources based on the magnitude of closeness $cl_i^+$.
12.   IF ($cl_i^+ > cl_j^+$) THEN
          $R_i$ is preferred to $R_j$.
13.   ENDIF
14. END ResourceSelection
```

Figure 2. Resource selection algorithm.

B. Construct a Decision Matrix D

In the context of resource selection, the effect of each criterion cannot be considered alone and should be viewed as a trade-off with respect to other criteria. The decision matrix, D can be constructed as follows:

$$D = \begin{array}{c c} & \begin{array}{cccc} C_1 & C_2 & \cdots & C_n \end{array} \\ \begin{array}{c} R_1 \\ R_2 \\ \vdots \\ R_m \end{array} & \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \cdots & f_{mn} \end{bmatrix} \end{array} \quad (4)$$

with $W = (w_1, w_2, \ldots, w_n)$, where w_j is the weight of the criterion C_j satisfying $\sum_{j=1}^{n} w_j = 1$. R_i denotes the alternative resources i, i=1, 2,…, m; C_j represents the j^{th} criterion, j = 1, 2, … , n related to i^{th} resource, and f_{ij} is a crisp value indicating the performance value of for each resource R_i with respect to each criterion C_j.

```
Algorithm WeightAssignment
INPUT: $freshness_{R_i}$, $efficiency_{R_i}$, $reliability_{R_i}$
BEGIN
1.  FOR each criterion DO
2.      Normalize D for each criterion as
            $p_{ij} = f_{ij} / (\sum_{i=1}^{m} f_{ij})$, with $j \in [1,\ldots,m]$.
3.      Determine the entropy $E_j$ of the set of normalized outcomes of criterion j based on the equation:
            $E_j = -\varphi \sum_{i=1}^{m} p_{ij} \ln p_{ij}$ ,where $\varphi = 1/\ln(m)$ which guarantees $0 \leq E_j \leq 1$.
```

```
4.    Assign weight for criterion $j$ with $w_j = d_j / (\sum_{i=1}^{n} d_i)$
      where $d_j = 1 - E_j$.
5. ENDFOR
END WeightAssignment
```

Figure 3. Entropy-based weight assignment algorithm.

C. Weight Assignment

Our proposed resource selection algorithm exploits the TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) method [10] where it requires a decision matrix as input data and uses given relative weights to drive the optimization process. However, with respect to our research, the disadvantage of TOPSIS method is that the weights are entered manually. Therefore, we attempt to overcome this disadvantage by using the entropy method for weight assignment as shown in Fig. 3.

V. Experimental Configuration

In order to evaluate the effectiveness of the proposed resource selection algorithm, we undertook an extensive simulation experiments. We simulated up to 200 resource nodes with different number of workload, i.e. 100, 500 and 1000. The value range of resource storage capacity and its processing speed together with network bandwidth considered are based on research in [12] and [13].

We assume that job arrival time is a Poisson process and jobs processing time follow exponential distribution. Size of job is determined based on TPC-W benchmark database size [13], [14] which is ranged from 10MB to 800MB while resource storage capacity is 10GB. Meanwhile, resource processing speed is set to range from 20MB/s up to 1000MB/s. The parameters setting in the simulation are summed up as follows:

- Resource processing speed: 20MB/s, 100MB/s, 300MB/s, 500MB/s, 750MB/s, 1000MB/s.
- Network bandwidth: 10MB/s, 45MB/s, 155MB/s, 1GB/s, 2.5GB/s, 10GB/s.
- Size of job: 10MB, 200MB, 400MB, 600MB, 800MB.
- Reliability value: ranged between [0, 1).

All simulations are conducted based on three different workload characteristics, i.e. high, medium and low conflict rates as shown in Table I.

TABLE I. Workload Tested (Simulation Scenarios)

Scenario	Conflict Rate	Job Probability		
		$Total_{op}^{sys}$	$Causal_{op}^{sys}$	$Comm_{op}^{sys}$
1	High	0.45	0.45	0.10
2	Medium	0.30	0.30	0.30
3	Low	0.10	0.10	0.80

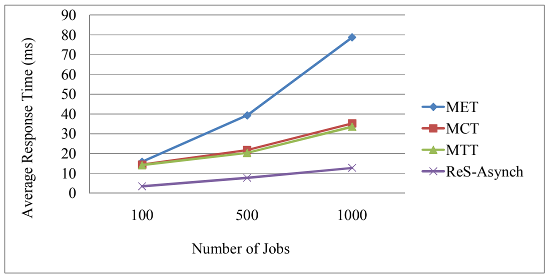
VI. Results and Discussions

The performance evaluation is done based on *Average Response Time* (*ART*) which is defined as:

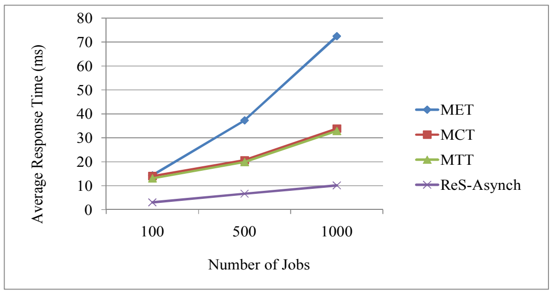
$$ART = (\sum_{i=1}^{n} RT_{u_i}) / n \qquad (5)$$

where RT_{u_i} is response time for job u_i with n is the total number of jobs. We compare our proposed algorithm against three existing resource selection heuristics. Two of them, namely Minimum Execution Time (MET) and Minimum Completion Time (MCT) algorithms are most recently used in [8]. An additional algorithm called Minimum Transfer Time (MTT) algorithm is also implemented in our simulation. MTT algorithm is similar to MCT algorithm in the sense that it finds the best resource which can process job with shortest completion time, but with additional consideration of highest network link capacity for minimum job propagation time.

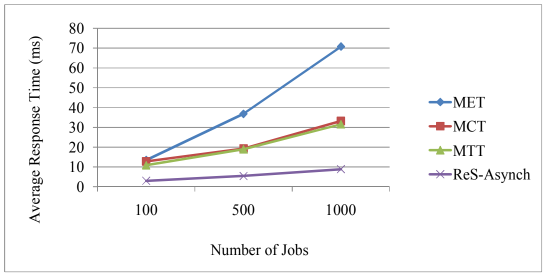
Note that the above mentioned heuristics do not consider an asynchronous replication (i.e. it neglects the risk of conflicting jobs) in their selection techniques. Also, these algorithms do not consider the reliability factor which is likely inevitable in highly distributed and heterogeneous environment. However, these heuristics are implemented in our evaluation study since none of the existing algorithm can be directly comparable to our algorithm. In our simulation, MET, MCT and MTT algorithms are implemented with the underlying asynchronous replication (based on UO approach) environment. That is, the selected resource based on MET, MCT and MTT algorithms has to comply with consistency requirement during job execution and propagation process.



(a)



(b)



(c)

Figure 4. An ART with respect to different scenarios. (a) Scenario 1 – high conflict rate (b) Scenario 2 – medium conflict rate (c) Scenario 3 – low conflict rate.

Figs. 4(a), 4(b) and 4(c) show the ART for scenario 1, 2 and 3 respectively. Meanwhile, Table II summarizes the entire comparative results for all scenarios with N1, N2 and N3 denotes 100, 500 and 1000 number of jobs respectively. Clearly, the significance of our proposed algorithm (i.e. ReS-Asynch) is verified in these results. The experiments show that our algorithm outperforms the MET, MCT and MTT algorithms with significant low response time for all type of workloads i.e. high, medium and low conflict rate.

Overall, our algorithm achieved less than 13ms response time for all workload tested. In contrast, MET algorithm reached up to more than 78ms. Although the MCT and MTT heuristics can provide a lower value of job response time as compared to MET algorithm, these heuristics neglect the factor of data freshness and resource reliability, which makes the MCT and MTT heuristics produced longer job response time as compared to our algorithm. A significant small value of ART achieved by our algorithm is due to the consideration of two most important characteristics of distributed, asynchronous replicated environment, i.e. degree of data freshness and reliability value of resources.

TABLE II. OVERALL COMPARATIVE RESULTS OF ART FOR SCENARIO 1, 2 AND 3

Algorithm	**Scenario1**			**Scenario 2**			**Scenario 3**		
	N1	N2	N3	N1	N2	N3	N1	N2	N3
MET	15.9	39.3	78.7	14.4	37.3	72.5	13.4	36.8	70.8
MCT	14.4	21.8	35.3	14.0	20.6	33.9	12.8	19.4	33.2
MTT	14.2	20.4	33.6	13.1	19.9	32.9	10.9	18.9	31.6
ReS-Asynch	**3.6**	**7.8**	**12.8**	**3.1**	**6.7**	**10.1**	**3.0**	**5.4**	**8.9**

VII. CONCLUSION AND FUTURE WORK

This paper addressed the problem of resource selection for applications that update data in enterprise grid systems. The algorithm first determines different ***QoS*** (i.e., freshness, efficiency and reliability) values for each resource. It then constructs a decision matrix *D* (step 4) and assigns weights to freshness, efficiency and reliability based on entropy. We have shown that our simple, yet effective algorithm selects the best available resources which can provide the shortest job response time. In future, we plan to investigate the impact of the proposed approach on energy efficiency in enterprise grid resources.

REFERENCES

[1] J. H. Abawajy, "Adaptive hierarchical scheduling policy for enterprise grid computing systems," *Journal of Network and Computer Applications,* vol. 32, pp. 770-779, 2009.

[2] P. Strong, "Enterprise Grid Computing." vol. 3: ACMQueue, 2005.

[3] B. Javadi, M. K. Akbari, and J. H. Abawajy, "Analytical communication networks model for enterprise Grid computing," *Future Generation Computer Systems,* vol. 23, pp. 737-747, 2007.

[4] N. Lilian Noronha, Jos, N. Marcos, Fl, V. vio, and A. cius de, "Resource selection in grid: a taxonomy and a new system based on decision theory, case-based reasoning, and fine-grain policies," *Concurr. Comput. : Pract. Exper.,* vol. 21, pp. 337-355, 2009.

[5] Z.-j. Li, C.-t. Cheng, and F.-x. Huang, "Utility-driven solution for optimal resource allocation in computational grid," *Computer Languages, Systems & Structures,* vol. 35, pp. 406-421, 2009.

[6] E. Elmroth and J. Tordsson, "Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions," *Future Generation Computer Systems,* vol. 24, pp. 585-593, 2008.

[7] L. M. Khanli and M. Analoui, "An approach to grid resource selection and fault management based on ECA rules," *Future Generation Computer Systems,* vol. 24, pp. 296-316, 2008.

[8] C.-M. Wang, H.-M. Chen, C.-C. Hsu, and J. Lee, "Dynamic resource selection heuristics for a non-reserved bidding-based Grid environment," *Future Generation Computer Systems,* vol. 26, pp. 183-197, 2010.

[9] W. Zhou, L. Wang, and W. Jia, "An analysis of update ordering in distributed replication systems," *Future Generation Computer Systems,* vol. 20, pp. 565-590, 2004.

[10] P. Sen and J.-B. Yang, *Multiple criteria decision support in engineering design*. New York: Springer-Verlag Berlin Heidelberg, 1998.

[11] U. T. Mattsson, "Database Encryption - How to Balance Security with Performance," *SSRN eLibrary,* 2005.

[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, and P. Pat, "VL2: A Scalable and Flexible Data Center Network," in *SIGCOMM*, 2009.

[13] Z. Xi, R. Alma, and R. Erik, "Characterization of the E-commerce Storage Subsystem Workload," in *Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*: IEEE Computer Society, 2008.

[14] K. Manassiev and C. Amza, "Scaling and Continuous Availability in Database Server Clusters through Multiversion Replication," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007. DSN '07*, 2007, pp. 666-676.