International Journal on Advanced Science Engineering Information Technology

Vol.7 (2017) No. 2
ISSN: 2088-5334

Cost and Performance-Based Resource Selection Scheme for Asynchronous Replicated System in Utility-Based Computing Environment

Wan Nor Shuhadah Wan Nik[*], Bing Bing Zhou[#], Jemal H. Abawajy[&], Albert Y. Zomaya[#]

[*]*Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Terengganu, Malaysia*
E-mail: wnshuhadah@unisza.edu.my

[#]*School of Information Technologies, University of Sydney, Australia*
E-mail: bing.zhou@sydney.edu.au, albert.zomaya@sydney.edu.au

[&]*Faculty of Science, Engineering and Built Environment, Deakin University, Australia*
E-mail: jemal.abawajy@deakin.edu.au

***Abstract*— A resource selection problem for asynchronous replicated systems in utility-based computing environment is addressed in this paper. The needs for a special attention on this problem lies on the fact that most of the existing replication scheme in this computing system whether implicitly support synchronous replication and/or only consider read-only job. The problem is undoubtedly complex to be solved as two main issues need to be concerned simultaneously, i.e. 1) the difficulty on predicting the performance of the resources in terms of job response time, and 2) an efficient mechanism must be employed in order to measure the trade-off between the performance and the monetary cost incurred on resources so that minimum cost is preserved while providing low job response time. Therefore, a simple yet efficient algorithm that deals with the complexity of resource selection problem in utility-based computing systems is proposed in this paper. The problem is formulated as a Multi Criteria Decision Making (MCDM) problem. The advantages of the algorithm are two-folds. On one fold, it 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. On the other fold, this representation further relaxed the complexity in measuring the trade-offs between the performance and the monetary cost incurred on resources. The experiments proved that our proposed resource selection scheme achieves an appealing result with good system performance and low monetary cost as compared to existing algorithms.**

***Keywords*— resource selection; resource management; utility-based computing; grid/cloud computing; multi criteria decision making (MCDM) method; asynchronous replication**

I. INTRODUCTION

Undoubtedly, both Grid and Cloud computing share many characteristics as both emerged from the "computing as a utility" paradigm [1]. Enterprise Grid and Cloud computing are the two most touted utility-based computing environments in recent years [2]-[7]. Both systems share two common characteristics; 1) data is replicated and may reside in distributed resources in order to achieve high data availability and fault-tolerance and to enhance performance; 2) they are both business-oriented and may run an update-intensive applications (e.g. online shopping, e-ticketing, etc.). In such situations, the challenges faced by these computing systems are realized when both systems have to deal with the complexity of data management in the presence of jobs that update data. Therefore, the employment of an efficient data management and replication scheme is paramount in this kind of system. Generally, the two most common data replication schemes in distributed systems are the synchronous and asynchronous replications. The implementation of synchronous replication is very expensive since complex protocols are required to ensure serializability [8], [9]. Hence, this study employs an asynchronous replication, which is desirable for data with weaker consistency requirements to achieve better performance. More specifically, an asynchronous replication scheme called Update Ordering (UO) approach [10] is exploited in this research work.

Both Enterprise Grid and Cloud computing systems can also be characterized in terms of the scale and characteristic

of their resources. Cloud computing is well-known as a large-scale distributed computing environment that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet [11]. Meanwhile, from the perspective of Enterprise Grid system, although the number of resources in this class of system is relatively small as compared to traditional Grid applications (e.g., scientific domain), during seasonal or unexpected spikes in demand for a product retailed by a very large corporation, a significant number of additional resources may become involved and need to be allocated to satisfy user requests. In such a scenario, resource selection for job processing becomes a major problem for both Enterprise Grid and Cloud computing systems. Further, the problem becomes harder when both performance and cost are the major concern in the resource selection process in order to find the best resources to run the job in low response time with minimum monetary cost.

Therefore, by taking into consideration the above mentioned issues, the work done in this paper focuses on three key points: (1) how to implement an efficient asynchronous replication scheme in a utility-based computing environment, (2) how to design a system model which can intelligently represents the important, interrelated and interdependent components that comprise the utility-based computing system so that applications that update data can be run efficiently in this system environment, which further supports the development of an effective and efficient resource selection scheme, and (3) How to select the best resources for jobs in an application that employs asynchronous replication in a utility-based computing environment so that the performance of the replicated system (in terms of job response time) is high while preserving minimum monetary cost incurred on distributed resources.

With regard to the first point, this paper first presents its first contribution by providing a detailed discussion of the Update Ordering (UO) approach and shows how the exploitation of this approach is done in our study to support the development of effective solutions for the resource selection problem in a utility-based computing environment. Meanwhile, with regard to the second point which reflects the second contribution of this paper, a new high-level system framework for applications that update data in utility-based computing environments is developed. More specifically, the functionalities of the important system components are discussed incorporation with the implementation of the UO approach. The framework is designed in such a way that it can be adopted and implemented in both Enterprise Grid and Cloud computing systems. Further, to respond to the third point, two different issues must be addressed simultaneously in order to solve the problem: (1) one should first deal with the difficulty of predicting the performance of resources in terms of job response time, and (2) an efficient mechanism must be employed in order to measure the trade-off between the performance and the monetary cost incurred on resources so that minimum cost is preserved while providing low job response time.

To deal with the first issue, we first define job response time as the time elapsed between job arrivals at the resource until the execution result is returned to the user. When a job that updates data is present, resource is needed to propagate its update value to other resources in order to maintain data consistency. However, to avoid conflicts (data inconsistencies), the resource is only allowed to process the job when it gets the latest value of data. Indeed, the arrival of update propagation from other resources is unpredictable due to the dynamicity and heterogeneity of the distributed systems, which makes the estimation of job response time very difficult. Further, when the second issue is put into focus, the development of an efficient mechanism is indeed required so that it does not put an excessive burden on the complexity of the first issue but at the same time can ensure job requests can be processed in low job response time with minimum monetary cost.

By considering both issues simultaneously, the paper further presents its third contributions by providing a new prediction technique on resource performance with respect to job response time. That is, several QoS criteria, namely, efficiency, freshness, and reliability are defined in such a way that each of these criteria will reflect important factors that affect job response time in an asynchronously replicated system. Finally, as a fourth contribution, we solved the problem of selecting the best resource which can process jobs in the shortest time by addressing the problem as a Multi Criteria Decision Making (MCDM) problem, which further evaluates the trade-offs between these criteria (i.e., performance criteria) at each resource. This problem solving method is then exploited to settle the performance and monetary cost issue simultaneously. That is, the performance QoS criteria set-efficiency, freshness and reliability-previously considered is combined with the cost criterion in the evaluation process. This makes the overall criteria set efficiency, freshness, reliability and cost.

Particularly, in addition to the exploitation of UO approach as previously discussed, work in this paper also exploits the TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) method [12] where it is reconciled with an entropy method in a resource selection engine to allocate jobs to the best available resource to provide both low response time and low monetary cost. The TOPSIS method is one of the well-known MCDM techniques used in engineering fields. To the best of our knowledge, this study is the first attempt to consider an asynchronous replication in resource selection for a utility-based computing system, especially in Enterprise Grid and/or Cloud environments.

Compared to our previous work presented in [13], [14], this paper explore and take advantage of our proposed framework presented in [13] together with part of the resource selection algorithm presented in [14] in order to come out with the efficient resource selection scheme which consider both system performance and resource monetary cost in asynchronous replicated system. Specifically, we focus on how to solve the above-mentioned problems simultaneously. The performance of the proposed resource selection scheme is extensively studied, especially when all the above issues need to be addressed concurrently.

Related research can be viewed from three perspectives: (1) job scheduling, (2) replica/resource selection and (3) data replication and transaction management. However, most of the existing research on job scheduling either does not directly account for data replication, or it totally neglects the transaction management factor. Therefore, we focus our review of related works on replica/resource selection together with data replication and transaction management in Grid and/or Cloud environments.

In Grid environments, replica selection is usually studied within the area of replication management in Data Grids, where most replications are at file-level granularity. Moreover, replicated data are not allowed to be updated, or they require manual placement of the files [15]-[20]. Most of them deal with read-only jobs, very rarely with write jobs. In read-only applications, replica selection is usually decoupled with replica placement by considering such factors as access pattern and network latency and bandwidth and focus on reducing data access time [21]-[23]. Recently, work in [24] exploits data mining technique in order to improve the performance of data replication and replica selection in Data Grid system while research in [25] provide a comprehensive survey on data replication in grid computing environment. All these works assume read-only jobs on replicated data.

Indeed, replication management and updates have been put on the list of requirements in the area of Data Grids due to the emergence of Enterprise Grid applications [26]. To the best of our knowledge, none of the existing resource selection techniques in Grid environments provide a mechanism on how to select the best resource when jobs that update data are present. This lies on the fact that most of the recent surveys on data replication and replica selection in the area of Data Grids mostly focused on read-only jobs as in [17], [21], [27]. Further, a resource selection model based on Decision Theory is proposed in [28], [29]. These works are similar to our work in the sense that they apply a decision theory to select the 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.

As compared to Grid environments, the existence of both read-only and update data are considered by a number of research works in resource/replica selection for Cloud environments. Research in [30] is one of the earliest research efforts that measure the potential benefits of effective resource selection in Cloud environments. The authors mentioned that the problem of workload analysis [45] and resource selection had not been seriously addressed yet while the research work was done. Further recently, work in [31] characterizes workloads of Cloud system based on several performance parameters such as job turnaround time and throughput in order to achieve an optimal resource allocation. Notwithstanding this, the experiments were done with no specific consideration of the existence of update job in Cloud applications.

Further, work done in [32] presents a workload-aware approach to making the design decision of what data items to replicate and where to place the data items as well as replicas with the goal to minimize the resources consumed in executing the workload. The minimization of the resource usage is measured based on a parameter named Average Query Span, i.e. the average number of machines involved in the execution of a query or a transaction (OLTP). This parameter is an abstract performance metric, which is introduced to handle the difficulty of directly model the resource consumption of update jobs/tasks. In other words, the research specifically addressed two classes of workloads, i.e. analytical read-only workloads and OLTP workloads. This research is similar to our work as both research works put an effort in handling the complexity of resource/replica management when jobs that update data are considered. However, while the researchers in [32] focused on synchronous update, our work provides a solution for asynchronously replicated system.

In [33], 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 have previously been assigned to it (i.e., the execution time for all jobs in the queue at resource). 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. There is little work on mechanisms for best resource selection especially in Enterprise Grid and Cloud systems. To the best of our knowledge, our work is the first attempt that specifically focuses on providing a mechanism on how to select the best resources for applications that update data especially when an asynchronous replication is considered in both computing systems.

II. Material and Method

A. System Model

A 3-tier architecture of utility-based computing platform is shown in Fig. 1. Users at Tier-1 (user level) send a job request to the Resource Broker (RB) which located at Tier-2 (middleware level). Then, this RB works together with Transaction Service (TS) at the same tier to select and allocate a job to an appropriate resource for execution at Tier-3 (resource level). Each resource at Tier-3 is equipped with the Transaction Manager (TM). The communication between resources at this tier is established by TM at each resource.

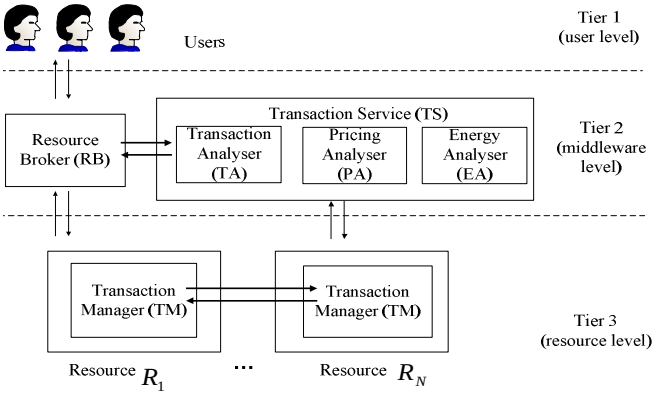


Fig. 1 3-tier architecture of utility-based computing platform [13]

By considering an Update Ordering (UO) [10] approach, we assume an n-replica system, $\Re^n = \{R_1, \cdots, R_n\}$. A job received by resource R_i directly from Resource Broker (RB) is said to have originated from R_i. When a request is received by a resource, it is stored in a buffer/log and waits to be checked on its ordering constraint (i.e. total, causal and commutative). Once its ordering constraint is satisfied, the job is executable or deliverable. In other words, that request is ready to be executed by the resources. The update-ordering constraint is defined as follows:

Definition 1: (FIFO ordering constraint " $\rightarrow$ "). If two updates, u_1 and u_2 are originated and sent to the replica group from the same replica R_i and if u_1 is delivered before u_2 at the original replica, then $u_1 \rightarrow u_2$ iff: u_1 is delivered before u_2 at the rest of the replicas.

Definition 2: (causal ordering constraint " $\prec$ "). If R_i delivers an update u_1 originated from R_j before sending out an update u_2, then $u_1 \prec u_2$, iff: u_1 is delivered before u_2 at all replicas.

Definition 3: (total ordering constraint " $\leftrightarrow$ "). For two updates u_1 and u_2 sent from R_i and R_j, then $u_1 \leftrightarrow u_2$, iff: When one replica delivers u_1 before u_2, the rest of the replicas deliver u_1 before u_2 as well. Or the other way around when one replica delivers u_2 before u_1, the rest of the replicas deliver u_2 before u_1 as well.

Definition 4: (total + causal ordering constraint " $\Rightarrow$ "). If two updates u_1 and u_2 are originated from R_i and R_j respectively, then $u_1 \Rightarrow u_2$, iff: $u_1 \prec u_2$ and $u_1 \leftrightarrow u_2$.

To decide the ordering constraint for each update operation, the inter-operation semantics between update operations needs to be analysed. The semantics is based on whether two update operations are commutative or not. Suppose that a service provides a set of update operations $OP_{srv} = \{u_1, u_2, \cdots, u_n\}$ and assume that u_1 and u_2 are any two update operations, u_1 and u_2 can be the same operation. Their operation semantics is defined to have the following two relations:

Definition 5: (commutative relation "||"). $u_1 \| u_2$ iff: the effect of executing (u_1, u_2) equals the effect of executing (u_2, u_1).

Definition 6: (conflicting relation " $\triangleright\triangleleft$ "). $u_1 \triangleright\triangleleft u_2$ iff: the effect of executing (u_1, u_2) is different from that of executing (u_2, u_1).

Therefore, the commutative operation is defined as follows:

Definition 7: (commutative operation). An update operation u is a commutative operation iff: $\forall v \in OP_{srv}$, $v \| u$.

This is to say if u is a commutative with every operation in OP_{srv}, u is a commutative operation. A commutative operation implies that the order of its execution does not affect the state of replicas.

Definition 8: (total operation). An update operation u is a total operation iff: $u \in OP_{srv}$ and $\exists v \in OP_{srv}$, $v \triangleright\triangleleft u$.

Definition 9: (caused-by relation " $\overset{r}{\prec}$ "). If $u_1 \overset{r}{\prec} u_2$, iff: $u_1 \prec u_2$ and u_2 is the real effect of executing u_1.

Definition 10: (caused-by operation). An update operation u is a caused-by operation, iff: $\exists v \in OP_{srv}$, $v \overset{r}{\prec} u$.

Therefore, the following three types of operation sets are defined:

Definition 11: (total job operation set – $Total_{op}$). $Total_{op}$ contains all total job operation out of OP_{srv}.

Definition 12: (commutative job operation set- $Comm_{op}$). $Comm_{op}$ contains all commutative job operations out of OP_{srv}.

Definition 13: (causal job operation set – $Causal_{op}$). $Causal_{op}$ contains all caused-by operations out of OP_{srv}.

Further, assume that 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

$$\left|\Re^k\right| = \left|\{s_1^k, s_2^k, ..., s_N^k\}\right| \quad (1)$$

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

$$\theta_{sys} = \{\{J^{d_1}\}, \{J^{d_2}\}, ...\{J^{d_m}\}\} \quad (2)$$

where $\{J^{d_k}\}$ is a set of job operations (i.e., total, causal and commutative [10]) for d_k with k=1, 2,…, m and

$J^{d_k} = \{u_1^{d_k}, u_2^{d_k}, ..., u_n^{d_k}\}$, n representing the total number of job operations on d_k such that $u_1^{d_k}$ and $u_2^{d_k}$ are job operations on the same data d_k .

For example, if job u^{d_k} is commutative with every job in J^{d_k} , then u^{d_k} is said to be 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 said to be 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 apply:

Definition 14: (total job operation set- $Total_{op}^{d_k}$). $Total_{op}^{d_k}$ contains all total job operations out of J^{d_k} .

Definition 15: (commutative job operation set- $Comm_{op}^{d_k}$). $Comm_{op}^{d_k}$ contains all commutative job operations out of J^{d_k} .

Definition 16: (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 with each other although they are conflicting in their own dataset since each of them is working on different data. With $Total_{op}^{sys}$, $Causal_{op}^{d_k}$ and $Comm_{op}^{sys}$ are the total number of total operation, causal operation and commutative operation in the system respectively. Therefore, we have the following

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

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

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

$Total_{op}^{d_k} \cap Total_{op}^{d_l} = \phi$, $Causal_{op}^{d_k} \cap Causal_{op}^{d_l} = \phi$ and $Comm_{op}^{d_k} \cap Comm_{op}^{d_l} = Comm_{op}^{sys}$.

B. Job Response Time in Asynchronous Replication System

This study considers the implementation of asynchronous replication, which employs asynchronous propagation strategy among resources that hold the replica data. In an asynchronous propagation, the update of job execution is propagated after the result is returned to the user. In contrast, synchronous propagation strategy sends an update of job execution to the peer replicas before the result is returned to the user. It is clear that asynchronous replication gives a better response time to the user. Therefore, in this study, job response time is defined as the time when a user submits a job request to the system until the user receives the result of job execution from the system. To give a clearer view of the definition of job response time considered in this study, we illustrate an example scenario for asynchronous update propagation strategy as shown in **Error! Reference source not found.**. 2. In this example, a job response time is defined as the time period between t_1 until t_6 . The time when a user submits a job request to the system is designated t_1 . Later, t_2 indicates the time when the RB receives the job request directly from a user before submitting it to the best resource (i.e., the originated resource, R_x) for job processing and execution at time t_3 . Meanwhile, the time period between t_3 and t_4 indicates the time period required to process the job request by originated resource R_x before the result is submitted and received by the RB at time t_5 . Finally, this result is returned and received by the user at time t_6 . Meanwhile, t_7 indicates the time when the update propagation process by originated resource R_x is started. It is the responsibility of R_x to propagate its update value to other peer resources that hold the same replica R_y where y = 1,2,…, n, $R_x \neq R_y$ and $R_x, R_y \in \Re^N$. The time when resource R_1 and R_n receive an update value from originated resource R_x is indicated with time t_8 and t_9 respectively.

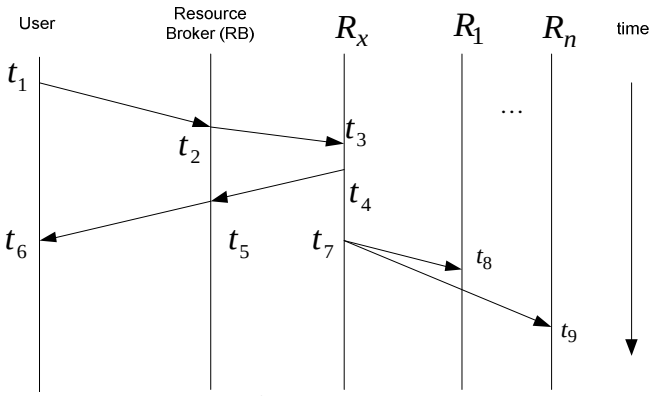


Fig. 2 A scenario of asynchronous propagation strategy

C. Multi-Criteria Decision Making (MCDM): Topsis Method

Generally, decision making involves the balancing of multiple, potentially conflicting requirements. Classical optimization deals with these problems by taking the most important requirement as the objective function and the remainder as constraints, which still leaves the problem of potentially irreconcilable requirements. The common approach employed to deal with this problem is to relax the thresholds of the constraints until feasible solutions emerge [34]. However, the Multi Criteria Decision Making (MCDM) method provides an alternative view on solving this problem. MCDM is an interactive method that deals with multiple criteria problems by employing a range of processes that clarify the consequences of the underlying trade-offs between criteria in configuring alternative solutions. In other words, the MCDM problem deals with the evaluation of a set of alternatives in terms of a set of decision criteria.

The MCDM method is widely adopted in engineering projects for optimal decision making [34], [35]. There is a wide range of MCDM methodologies for choosing between alternative solutions that are available for the decision maker, and one of the well-known methods is TOPSIS [12]. This method is based on the concept that the chosen alternative should have the shortest distance to the ideal point and the furthest distance from the negative ideal point. These ideal points can be considered as dummy alternatives where resources are compared. The method requires a decision matrix for input evaluation data but uses given relative weights as the representation of preference information. The advantage of the TOPSIS method is that it can produce a clear preference order of a set of competing alternatives. The exploitation of TOPSIS in our work provides an effective mechanism for finding the best resource, the best alternative solution, to run the job because it can efficiently evaluate the trade-offs between all the QoS criteria considered-efficiency, freshness, reliability, and cost-on each resource. The discussion of this exploitation process is provided in the following section.

D. Development of Resource Selection Algorithm

In this section, we will present the proposed resource selection algorithm. Our proposed model can be divided into two subsections. Detailed definition and evaluation of each QoS criterion considered are provided in the first subsection. The direct implication of both the heterogeneity of network latencies and computational speed of resources on the estimation of job response time are represented in terms of freshness, efficiency and reliability value (i.e., performance QoS criteria). Meanwhile, cost criterion is defined to reflect monetary cost which may be incurred when a job is executed at any particular resource. Further, the next subsection presents the exploitation of TOPSIS and the entropy method and shows how it serves as a selection engine for all QoS criteria considered. Before we present the proposed algorithm, we will define some concepts used in the algorithm.

E. Defining QoS Criterion

For any particular resource R_x, the execution time for job u_i on this resource is defined as follows

$$(ET_{u_i})_{R_x} = size(u_i) / cs_{R_x} \quad (3)$$

where $size(u_i)$ is the size of the job $u_i \in \theta_{sys}$ and cs_{R_x} is the computing speed of resource R_x. In other words, it is the time taken by resource R_x with computing speed cs_{R_x} to execute an individual job u_i, where $size(u_i)$ is a size of the job $u_i \in \theta_{sys}$. Further, the efficiency value for any resource R_x, called $efficiency_{R_x}$, is defined as

$$efficiency_{R_x} = \sum_{i=1}^{n} (ET_{u_i})_{R_x} \quad (4)$$

where n is the job queue size in $readyQ_{R_x}$. That is, it measures how long a job is expected to wait before being executed (i.e., all jobs queued in $readyQ_{R_x}$ need to be completed), which indicates how busy R_x is in executing jobs in $readyQ_{R_x}$.

However, in real applications of distributed systems, the value of $efficiency_{R_x}$ is greatly affected by other delays which are very hard to measure accurately. The complex interaction between heterogeneous resources which may occur at different levels of distributed systems (e.g., physical resource level and network link level) becomes the main factor contributing to this difficulty. To overcome this problem and provide an efficient scheme for predicting the value of this delay, we introduce two other variables, freshness and reliability. At any particular time t, the freshness value for resource R_x namely $freshness_{R_x}$ is defined as follows

$$freshness_{R_x} = (\alpha - \beta)_{R_x} \quad (5)$$

where α is the ID (i.e., USN) of a job that is last originated in the system and β is the largest ID of a job in $readyQ_{R_x}$ on resource R_x for any particular data d_k, respectively. The freshness value on resource R_x indicates how likely it is that the new job will be put into $waitQ_{R_x}$ to avoid data inconsistencies. That is, it can be estimated by how many jobs are not yet being processed, which allows the prediction of how long the new job needs to wait before being processed in order to avoid conflicts. In this case, resource R_x is said to be the freshest resource in the system if $\alpha = \beta$. In other words, the resource R_x is said to be the freshest resource if $freshness_{R_x} = 0$. Meanwhile, the reliability for resource R_x called $reliabilit\,y_{R_x}$ is defined as

$$reliabilit\,y_{R_x} = (ART - ERT)_{R_x} \quad (6)$$

where ART and ERT are the averages of actual and estimated response time at resource R_X respectively with $ART \geq ERT$, $ART \neq 0$ and $ERT \neq 0$. This criterion captures other arbitrary delays which may occur in a distributed system not yet captured by $efficiency_{R_x}$ and/or $freshness_{R_x}$. As previously mentioned, these delays may be caused by complex interactions between heterogeneous resource hardware, software, networking and security [36]. It can be understood that the reliability criterion is used to measure the performance robustness of a resource. Therefore, from all the above definitions, it is realized that the above parameters or criteria are sufficient to measure the performance of each resource in selecting the best resource for job execution.

Further, the monetary cost criterion represents the service price specified by the resource owner based on a Service Level Agreement (SLA). The pricing mechanism decides how service requests processed by a resource are charged. Requests can be charged based on submission time (peak/off peak), pricing rates (fixed/changing) or availability of resources (supply/demand) [4]. Also, the pricing method may also depend on geographical locations [37] because pricing rates for resources at different geographical locations may vary significantly based on applicable taxes, fees or similar governmental changes of the resource location. Therefore, the price or servicing of a request by resources may vary over time and between different resources. Note that this criterion can also be considered as a monetary cost induced by energy cost on various resources [38]. Energy cost may significantly vary among resources based on current utilization or the geographical location of the resources, which may have different energy rates.

Indeed, it is difficult to determine the range of monetary costs at different resources when different resource providers are considered. Different providers may charge different rates depending on such factors as geographical rate and supply/demand rate. Federated Cloud [39] architecture, for example, provides a number of resources from different providers; these are transparently integrated to serve job requests by users. To deal with the complexity of various pricing rates among Cloud providers, we thus make an assumption that the rate of monetary cost for resources is scaled from 1 to10, where smaller values indicate lower pricing of resources. In other words, the input monetary value in the system is normalized to the homogeneous pricing rates. For example, a resource with the price rate of $0.22/machine hour can be normalized to 10 while the resource with the price rate of $0.11/machine hour can be normalized to 5.

The input monetary cost value is done by Pricing Analyser (PA) as shown in

. This analyser measures the resource price to run the job. It can be considered to be part of the SLA resource allocator component in the Cloud architecture as proposed in [4]. It is responsible for finding the potential cost reduction of providers, which could lead to a more competitive market and thus lowering pricing cost.

Based on the definition of each QoS criterion (efficiency, freshness, reliability and cost) presented previously, a lower value is more desirable as it can contribute to shorter job response time. Particularly, a lower freshness value indicates that the resource will be considered to hold fresher data in the system. A lower efficiency value shows faster job execution time by the resource. Lower reliability indicates that the resource is more resistant to the overhead caused by complex interactions among heterogeneous resources, while low cost value clearly shows the minimum monetary charges on resources for job execution. To further illustrate the complexity of the resource selection process with regard to these QoS criteria, we provide an example of possible scenarios where the user (or Resource Broker) needs to choose the best resource to process job requests with minimum response time and at low monetary cost, as shown in **Error! Reference source not found.**1. In this example, it is evident that resource 4 has the lowest reliability but has the worst efficiency value, which means that this resource should wait for processing a large number of jobs waiting in its queue. This is just a simple example that considers only 5 resources. One can imagine what would happen when the number of resources gets larger in real applications. We can expect that the complexity grows significantly as the number of resources grows.

TABLE I
AN EXAMPLE OF CRITERIA SET VALUE FOR RESOURCES

Resource ID	Freshness	Efficiency	Reliability	Cost
1	10	3.4	0.70	3
2	6	2.7	0.40	9
3	10	5.6	0.37	2
4	15	6.4	0.25	7
5	9	4.3	0.79	5

F. Resource Selection Algorithm

Fig. 3 shows the proposed resource selection algorithm. The algorithm first determines different QoS (freshness, efficiency, reliability, and cost) values for each resource (step 2). It then constructs a decision matrix D (step 4) and assigns weights to freshness, efficiency, reliability and cost based on entropy (step 5).

In the context of resource selection, the effect of each criterion (attribute) cannot be considered alone and always should be viewed as a trade-off with respect to other criteria. Any changes in, for instance, freshness, efficiency, reliability and cost may change the resource priorities. With this in mind, the ideal-point-based approach such as TOPSIS [12] seems to be a suitable framework for method selection problems since it allows explicit trade-offs and interactions among different criteria. Therefore, we exploit the TOPSIS method in order to find the best resources to run the job in minimum response time with a low monetary cost. The basic idea of TOPSIS is based on the concept that the chosen alternative should have the shortest distance to the ideal point and the furthest distance from the negative ideal point. This method requires a decision matrix as input evaluation data but uses given relative weights as the representation of preference information.

Algorithm ResourceSelection
BEGIN

1. **FOR** each resource $R_i \in \Re^N$ **DO**
2. Evaluate $freshness_{R_i}$, $efficiency_{R_i}$, $reliabilit\ y_{R_i}$ and $cost_{R_i}$
3. **ENDFOR**
4. Construct a decision matrix D
5. WeightAssignment ($freshness_{R_i}$, $efficiency_{R_i}$, $reliabilit\ y_{R_i}$, $cost_{R_i}$)
6. Normalize D and its weight whose elements are defined by $z_{ij} = f_{ij} / \sqrt{\sum_{i=1}^{m} f_{ij}^2}$, i=1,…, m; j=1,…,n.
7. Formulate the weighted normalized decision matrix whose elements are $x_{ij} = w_j z_{ij}$, i=1,…, m; j=1,…,n.
8. Determine the positive (a^+) and the negative (a^-) ideal solutions as follows:

$$a^+ = \{(\min_i x_{ij} \mid j \in J) \mid i = 1,..., m\} = \{x_1^+, x_2^+, ..., x_n^+\}$$

$$a^- = \{(\max_i x_{ij} \mid j \in J) \mid i = 1,..., m\} = \{x_1^-, x_2^-, ..., x_n^-\}$$

9. Calculate the separation measures for ideal and negative ideal solutions of each resources as follows:

$$R_i^+ = \sqrt{\sum_{j=1}^{n} (x_{ij} - x_j^+)^2} \text{ , i =1, …, m}$$

$$R_i^- = \sqrt{\sum_{j=1}^{n} (x_{ij} - x_j^-)^2} \text{ , i =1,…, m}$$

10. Calculate relative closeness of each resource to the ideal point as follows:

$cl_i^+ = R_i^- / (R_i^- + R_i^+)$, $0 \le cl_i^+ \le 1$ and i=1,…,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

Fig. 3 Resource selection algorithm

Suppose an MCDM problem can be represented as the example provided in **Error! Reference source not found.**1. Based on the resource selection algorithm presented in

Fig. 3 , the algorithm starts with the evaluation of each QoS criterion considered (i.e., freshness, efficiency, reliability, and cost). Then, in step 4, the decision matrix D is constructed as follows:

$$D = \begin{array}{c} \\ R_1 \\ R_2 \\ \vdots \\ R_m \end{array} \begin{array}{c} \begin{array}{cccc} C_1 & C_2 & \cdots & C_n \end{array} \\ \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \cdots & f_{mn} \end{bmatrix} \end{array} \quad (7)$$

with $W = (w_1, w_2, ..., w_n)$ 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 . With respect to our research, the disadvantage of the TOPSIS method is that the weights are entered manually. Indeed, human interventions can lead to errors and also do not support the dynamic nature of autonomous Grid and Cloud systems. Therefore, we attempt to overcome this disadvantage by using the entropy method for weight assignment (step 5).

The main advantage of the entropy method is that it eliminates the possibly biased judgment if the weights are assigned manually by the user. Based on the decision matrix D presented in Equation (7), its input data f_{ij} have different dimensions, and thus it needs to be normalized in order to transform the various criteria dimensions into the non-dimensional data, which allows comparison across all the criteria. We refer to the weight assignment algorithm based on the entropy method as shown in Fig. 4 . In this algorithm, matrix D is normalized for each criterion C_j (step 2 in Fig. 4). Further, the entropy E of the set of normalized outcomes of criterion j is determined based on the equation shown in step 3. Therefore, the best weight is given by the equation shown in step 4.

Algorithm WeightAssignment
INPUT: $freshness_{R_i}$, $efficiency_{R_i}$, $reliabilit\ y_{R_i}$, $cost_{R_i}$
BEGIN

1. **FOR** each criterion **DO**
2. Normalize D for each criterion as $p_{ij} = f_{ij} / (\sum_{i=1}^{m} f_{ij})$, with $j \in [1,...,m]$.
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 $0 \le E_j \le 1$.

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

Fig. 4 Weight assignment algorithm

Then, we refer back to the resource selection algorithm in Fig. 3 , where the normalized decision matrix is constructed whose elements are defined as in step 6. Consequently, each attribute has the same unit length of vector. Later, the weighted normalized decision matrix is formulated; its elements are given in step 7. The main idea of the TOPSIS approach is shown in steps 8-10. In step 8, the ideal point a^+ and negative ideal point a^- first need to be defined. Here, a^+ and a^- act as dummy alternatives (resources) used as a reference so that all alternatives (resources) can be properly evaluated as they represent the notional "best" and "worst" resource respectively. Referring to the example presented in **Error! Reference source not found.**1, the possible value of a^+ and a^- is shown in **Error! Reference source not found.**2. Later, the separation measures for ideal and negative ideal solutions for each resource are calculated in step 9. Then, the relative closeness of each resource to the ideal point is calculated using the equation shown in step 10. Finally, each resource is ranked based on the magnitude of closeness, cl_i^+ (steps 11 and 12). Indeed, the main advantage of the TOPSIS method is that it can produce a clear preference order or set of competing alternatives, in our case, the resources.

TABLE III
AN EXAMPLE OF CRITERIA SET VALUE FOR RESOURCES BY CONSIDERING DUMMY RESOURCES

Resource ID	Freshness	Efficiency	Reliability	Cost
1	10	3.4	0.70	3
2	6	2.7	0.40	9
3	10	5.6	0.37	2
4	15	6.4	0.25	7
5	9	4.3	0.79	5
a^+	**6**	**2.7**	**0.25**	**2**
a^-	**15**	**6.4**	**0.79**	**9**

G. Experimental Configuration and Evaluation Metrics

In this section, we present the experimental configuration and the performance metrics used to evaluate the effectiveness and efficiency of the proposed algorithm. We first present the experimental configuration. Later, we present the evaluation metrics.

H. Experimental Configuration

In order to evaluate the effectiveness of the proposed resource selection algorithm, we undertook an extensive simulation experiment. We simulated up to 50 resource sites with different numbers of workload, 1000, 2000, 3000, 4000 and 5000. The value range of resource storage capacity and its processing speed together with network bandwidth considered are based on research in [40], [41]. We assume that job arrival time is a Poisson process and job processing times follow an exponential distribution. The size of job is determined based on TPC-W benchmark database size [41], [42], which ranges 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 parameter settings 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: random value ranged between 0 and 1 second.
- Monetary cost value: normalized integer value ranged from 1 to 10.
- Number of jobs: 1000, 2000, 3000, 4000 and 5000 which are represented by N1, N2, N3, N4 and N5 respectively in **Error! Reference source not found.**4-**Error! Reference source not found.**6.

In this study, we also consider the situation where resource cost is not a relevant factor in the resource selection process. This is the case when resource pricing rates are the same for all resources in the system (e.g., available resources are in the same geographical edge or are on the same energy rate). In such a situation, our proposed algorithm will function as a performance prediction engine only by considering performance criteria (freshness, efficiency, and reliability) in the decision-making process without taking into consideration the monetary cost criterion. We summarize all experiment scenarios as shown in **Error! Reference source not found.**3. These experiments are based on two types of workload characteristics: conflict rate and monetary cost consideration. Accordingly, the experiments of high, medium and low conflict rates are done together with monetary cost either taken into consideration or not.

TABLE IIIII
WORKLOAD TESTED (SIMULATION SCENARIOS)

Scenario	Conflict Rate	Cost	Job Probability		
			$Total_{op}^{sys}$	$Causal_{op}^{sys}$	$Comm_{op}^{sys}$
1	High	No	0.45	0.45	0.10
2	Medium	No	0.30	0.30	0.40
3	Low	No	0.10	0.10	0.80
4	High	Yes	0.45	0.45	0.10
5	Medium	Yes	0.30	0.30	0.40
6	Low	Yes	0.10	0.10	0.80

I. Evaluation Metrics

In this study, we consider two evaluation metrics, Average Response Time (ART) and Average Monetary Cost (AMC). Both ART and AMC explicitly measure the performance of our proposed algorithm in terms of job response time and monetary cost respectively, which are defined as follows:

$$ART = (\sum_{i=1}^{n} RT_{u_i}) / n \quad (8)$$

$$AMC = (\sum_{i=1}^{n} MC_{u_i}) / n \quad (9)$$

where RT_{u_i} and MC_{u_i} are response time and monetary cost for job u_i respectively and n is the total number of originated jobs. Based on the above definitions, smaller values of ART and AMC are more desirable as they indicate a better system performance with lower monetary cost incurred on each job execution.

III. Results and Discussion

We compare our proposed algorithm against two other resource selection heuristics namely MCT and MCT_MTT algorithm. The MCT algorithm is initially discussed in [43] and later is used in [33]. This algorithm is the basic implementation of two well-known job scheduling heuristics: Min-min and Max-min [43]. In general, the fundamental concept of MCT is to assign jobs to a resource which can provide minimum completion time (MCT) defined as the earliest time that a resource can complete the execution of all jobs that have been previously assigned to it. Specifically, this algorithm tends to distribute the job to many available resources to reduce job queue length on each resource and to achieve the minimum completion time without any concern about conflicting jobs.

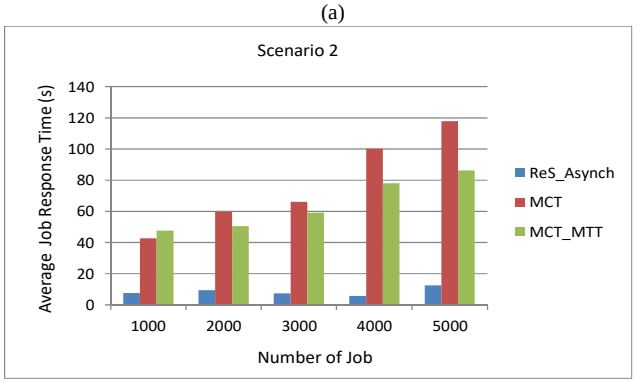
Another algorithm, MCT_MTT is developed as a variation of the MCT algorithm. MCT_MTT inherits the concept of MCT but with an additional consideration of minimum transfer time (MTT). The idea is adopted from the MTT algorithm proposed in [22], where the resource which has the fastest aggregate network bandwidth to other resources will be selected as the best resource to run the job. Therefore, the MCT_MTT algorithm selects the best resource that can provide both minimum completion time and minimum transfer time. The development of this algorithm allows us to investigate if the combination of considerations on network bandwidth capacity and completion time can achieve better performance than the original MCT.

In our simulation, MCT and MCT_MTT algorithms are implemented with monetary cost taken into account with the underlying asynchronous replication (based on UO approach) environment. That is the selected resource based on MCT and MCT_MTT algorithms will process a job request with a certain amount of monetary cost imposed on it, and at the same time comply with consistency constraint requirements during the job execution and job propagation phases.

A. An Evaluation of ART without Monetary Cost Consideration

Error! Reference source not found.(a)-5(c) show the ART for scenarios 1, 2 and 3 respectively. Meanwhile, **Error! Reference source not found.**4 summarizes the comparative results for all of these scenarios. In the absence of monetary cost consideration, the experiments show that our algorithm called ReS_Asynch algorithm outperforms the MCT and MCT_MTT algorithms with significant low response time for all types of workloads that is those with high, medium and low conflict rates.

(a)



(b)

(c)

Fig. 5 An ART with respect to different scenarios without monetary cost consideration. (a) Scenario 1-high conflict rate, (b) Scenario 2-medium conflict rate, (c) Scenario 3-low conflict rate

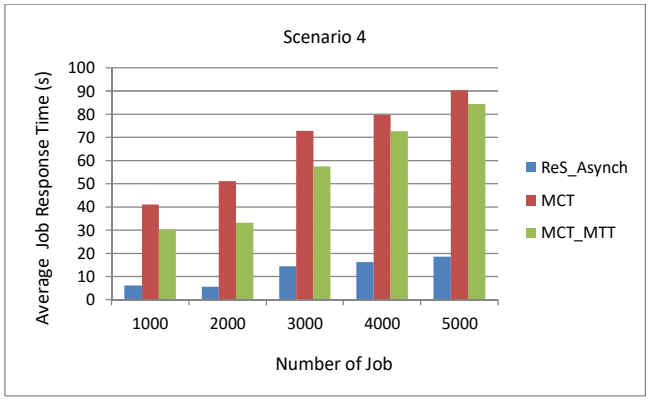
TABLE IVV
Overall Comparative Results of ART For Scenario 1, 2 and 3

Algorithm	**Scenario 1**				
	N1	N2	N3	N4	N5
ReS_Asynch	**6.57**	**9.26**	**5.58**	**12.30**	**11.88**
MCT	54.68	43.23	66.63	81.07	123.16
MCT_MTT	26.26	47.55	73.51	72.17	82.02
Algorithm	**Scenario 2**				
	N1	N2	N3	N4	N5
ReS_Asynch	**7.57**	**9.33**	**7.37**	**5.70**	**12.48**
MCT	42.69	59.83	66.13	100.23	117.91
MCT_MTT	47.69	50.57	59.16	78.06	86.28
Algorithm	**Scenario 3**				
	N1	N2	N3	N4	N5
ReS_Asynch	**3.20**	**6.00**	**8.68**	**8.30**	**16.66**
MCT	34.87	44.92	40.29	63.56	90.50
MCT_MTT	37.66	40.25	42.62	54.54	60.20

In the above results, an average job response time achieved by our algorithm (i.e., ReS_Asynch) is less than 17s for all workloads tested. In contrast, the MCT algorithm reached up to more than 123s. Meanwhile, an average job response time that achieved by the MCT_MTT algorithm ranges between 26.26s and more than 86s for all scenarios (scenarios 1-3).

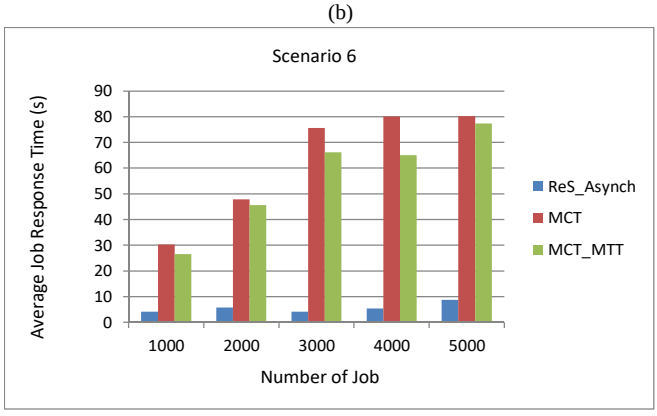
B. An Evaluation of ART with Monetary Cost Consideration

Here, **Error! Reference source not found.**(a)-6(c) show the ART when experiments are done for scenarios 4, 5 and 6 respectively. For more detail on the results obtained in these experiments, we summarize the entire comparative results for all of these scenarios in **Error! Reference source not found.**5.



(a)

(b)



(c)

Fig. 6 An ART with respect to different scenarios with monetary cost consideration. (a) Scenario 4-High conflict rate, (b) Scenario 5-Medium conflict rate, (c) Scenario 6-Low conflict rate

TABLE V
OVERALL COMPARATIVE RESULTS OF ART FOR SCENARIO 4, 5 AND 6

Algorithm	**Scenario 4**				
	N1	N2	N3	N4	N5
ReS_Asynch	**6.12**	**5.56**	**14.45**	**16.20**	**18.61**
MCT	41.13	51.12	72.78	79.69	90.40
MCT_MTT	30.34	33.20	57.54	72.74	84.44
Algorithm	**Scenario 5**				
	N1	N2	N3	N4	N5
ReS_Asynch	**4.09**	**5.01**	**7.38**	**8.32**	**16.13**
MCT	31.90	41.41	73.74	74.88	98.42
MCT_MTT	20.05	41.78	53.55	72.29	72.28
Algorithm	**Scenario 6**				
	N1	N2	N3	N4	N5
ReS_Asynch	**4.07**	**5.75**	**4.14**	**5.40**	**8.75**
MCT	30.33	47.80	75.68	80.20	80.24
MCT_MTT	26.55	45.56	66.22	65.07	77.38

Considering the monetary cost factor, we can expect that an ART achieved by ReS_Asynch will be slightly longer than in the previous experiment. This is because our algorithm has to tolerate the trade-offs between high performance and low monetary charges incurred on each job execution by resources. Notwithstanding this, the ReS_Asynch algorithm achieves more than one magnitude of order shorter response time as compared to the MCT algorithm for all number of jobs tested in a low-conflict-rate scenario. This promising result is especially realized in scenarios 5 and 6, where our proposed algorithm achieved as low as 4.09s and 4.07s of ART when 1000 number of jobs is involved in medium and low conflict rate respectively. On the other hand, regardless of any simulated scenario, the MCT and MCT_MTT algorithms only managed to achieve at the lowest of 30.33s and 20.05s of ART respectively. However, these values are still significantly high if compared to the achievements of our proposed algorithm.

C. An Evaluation of AMC

The experiment results of AMC gained in scenarios 4, 5 and 6 are visualized in **Error! Reference source not found.**(a)-7(c) respectively, while the value shown in **Error! Reference source not found.**6 summarizes the details of these results.

(a)

(b)

(c)

Fig. 7 An AMC with respect to different scenarios. (a) Scenario 4-High conflict rate, (b) Scenario 5-Medium conflict rate, (c) Scenario 6-Low conflict rate

TABLE VI
OVERALL COMPARATIVE RESULTS FOR AMC FOR SCENARIOS 4, 5 AND 6

Algorithm	**Scenario 4**				
	N1	N2	N3	N4	N5
ReS_Asynch	**2.00**	**1.00**	**2.00**	**1.00**	**1.00**
MCT	3.70	7.18	4.87	6.54	6.46
MCT_MTT	5.90	5.08	6.12	6.98	5.89
Algorithm	**Scenario 5**				
	N1	N2	N3	N4	N5
ReS_Asynch	**1.00**	**2.00**	**1.00**	**1.00**	**1.00**
MCT	4.60	5.70	6.98	6.31	6.21
MCT_MTT	2.70	8.80	7.78	3.50	9.61
Algorithm	**Scenario 6**				
	N1	N2	N3	N4	N5
ReS_Asynch	**2.00**	**2.00**	**2.00**	**3.00**	**2.00**
MCT	8.70	7.78	7.22	5.73	8.99
MCT_MTT	3.85	9.20	3.73	3.75	5.21

As expected, ReS_Asynch outperforms other existing algorithms in terms of monetary charge incurred on each job execution at resources. This is proven in the results shown above, where our proposed algorithm is managed to achieve the normalized value as low as 1.00 in most number of jobs tested in high and medium conflict rate scenarios. In clear contrast, all other existing algorithms achieve a very high normalized monetary cost value which ranges from 3.70 up to 9.61.

Obviously, the significance of our proposed algorithm is verified in all the results obtained from the experiments. From a performance point of view, although the MCT_MTT heuristic can provide a lower value of job response time as compared to the MCT algorithm in most scenarios tested, both heuristics neglect the factor of data freshness and resource reliability which makes them produce longer job response time as compared to our algorithm. Indeed, a significantly small value of ART achieved by our algorithm is due to the consideration of the two most important characteristics of distributed, asynchronous replicated environment: the degree of data freshness and the reliability value of resources. Furthermore, the ReS_Asynch algorithm is also able to provide minimum monetary cost incurred during job execution by resources as compared to other resource selection algorithms [44].

IV. CONCLUSION

This paper provides a new development of a simple yet efficient approach in dealing with the complexity of decision-making processes in update-intensive applications, particularly for an asynchronously replicated system in utility-based computing environments. To the best of our knowledge, the work in this paper is the first attempt. One part of this paper discusses the generalization of the design model that allows an easy adoption of this model in both Enterprise Grids and Cloud computing platforms. The development of this model provides an environment for an efficient implementation of an asynchronous replication scheme in a utility-based computing environment. Most importantly, this model includes and represents the important, interrelated and interdependent components that compose the utility-based computing systems so that applications that update data can be run efficiently in the targeted system environment, which further supports the development of effective and efficient resource selection schemes.

Further, the other part of this paper addresses the resource selection problem as a Multi Criteria Decision Making (MCDM) problem. The proposed framework hides the complexity of the resource selection process without neglecting important components that affect job response time. The difficulty in estimating job response time and its associated monetary cost trade-off is captured by representing them in terms of different QoS criteria levels at each resource. The experiments proved that our proposed algorithm achieves an appealing result with good system performance and low monetary cost as compared to existing algorithms. Most importantly, our simple yet effective framework proposed in this paper resolved the complexity of the prediction on update-job response time by representing it in terms of QoS.

ACKNOWLEDGMENT

This work is partially supported by Fundamental Research Grant Scheme (FRGS, Grant No: RR074) under the Ministry of Education (MOE) and Universiti Sultan Zainal Abidin (UniSZA), Malaysia. The authors are also grateful and wish to acknowledge the support of all members of Centre for Distributed and High Performance Computing at the University of Sydney and all staff of the Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin who have provided a vibrant and intellectually stimulating environment for this research.

REFERENCES

[1] D. G. Gomes, R. N. Calheiros, and R. T. Calasanz, "Introduction to the special issue on grid and cloud computing: Current advances and new research trends," *Computers and Electrical Engineering*, vol. 40, pp. 1634-1635, Jul. 2014.

[2] A. Patel, A. Seyfi, Y. Tew, and A. Jaradat, "Comparative study and review of grid, cloud, utility computing and software as a service for use by libraries," *Library Hi Tech News*, vol. 28. pp. 25-32, May 2011.

[3] M. G. Avram, "Advantages and challenges of adopting cloud computing from an enterprise perspective," *Procedia Technology*, vol. 12, pp. 529-534, Dec. 2014.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, pp. 599-616, Jun. 2009.

[5] M. Rahman, M. R. Hassan, and R. Buyya, "Jaccard index based availability prediction in enterprise grids," *Procedia Computer Science*, vol. 1, pp. 2707-2716, May 2010.

[6] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in *Proc. IEEE/ACM ICGC'09*, 2009, p. 50.

[7] W. Leesakul, P. Townend, P. Garraghan, and J. Xu, "Fault-tolerant dynamic deduplication for utility computing," in *Proc. IEEE ISO/C/SORTDC'14*, 2014, p. 397.

[8] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, "Consistency rationing in the cloud: Pay only when it matters," *Proceedings of the VLDB Endowment*, vol. 2, pp. 253-264, Aug. 2009.

[9] S. Abdi and S. Hashemi, "A hierarchical approach to improve job scheduling and data replication in data grid," *International Arab Journal of Information Technology*, vol. 12, pp. 278-285, May 2015.

[10] 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, May 2004.

[11] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. GCEW'08*, 2008, p. 1.

[12] L. N. Nassif, J. M. Nogueira, F. V. de Andrade, M. Ahmed, A. Karmouch, and R. Impey, "Job completion prediction in grid using distributed case-based reasoning," in *Proc. IEEE IWETICE'05*, 2005, p. 249.

[13] W. N. S. W. Nik, B. B. Zhou, A. Y. Zomaya, and J. H. Abawajy, "A framework for implementing asynchronous replication scheme in utility-based computing environment," in *Proc. ICCCBD'15*, 2015, p. 183.

[14] W. N. S. W. Nik, B. B. Zhou, A. Y. Zomaya, and J. H. Abawajy, "Efficient resource selection algorithm for enterprise grid systems," in *Proc. IEEE ISPDPA'11*, 2011, p. 57.

[15] M. Botón-Fernández, F. Prieto-Castrillo, and M. A. Vega-Rodríguez, "A self-adaptive resources selection model through a small-world based heuristic," *Journal of Supercomputing*, vol. 68, pp. 1441-1461, Jun. 2014.

[16] C. Li and L. Li, "A resource selection scheme for QoS satisfaction and load balancing in ad hoc grid," *Journal of Supercomputing*, vol. 59, pp. 499-525, Jan. 2012.

[17] T. Hamrouni, S. Slimani, and F. B. Charrada, "A survey of dynamic replication and replica selection strategies based on data mining techniques in data grids," *Engineering Applications of Artificial Intelligence*, vol. 48, pp. 140-158, Feb. 2016.

[18] S. Warhade, P. Dahiwale, and M. M. Raghuwanshi, "A dynamic data replication in grid system," *Procedia Computer Science*, vol. 78, pp. 537-543, Dec. 2016.

[19] R. J. Wilson, *The European DataGrid Project*, Institute de Fisica d'Altes Energies and Colorado State University, Barcelona, Spain and Fort Collins, Colorado, USA, 2001.

[20] M. Manohar, A. Chervenak, B. Clifford, and C. Kesselman, "Implementation and evaluation of a replicaset grid service," in *Proc. IEEE/ACM IWGC'04*, 2004, p. 218.

[21] R. K. Grace and R. Manimegalai, "Dynamic replica placement and selection strategies in data grids-A comprehensive survey," *Journal of Parallel and Distributed Computing*, vol. 74, pp. 2099-2108, Feb. 2014.

[22] R. M. Rahman, R. Alhajj, and K. Barker, "Replica selection strategies in data grid," *Journal of Parallel and Distributed Computing*, vol. 68, pp. 1561-1574, Dec. 2008.

[23] R. S. Chang and P. H. Chen, "Complete and fragmented replica selection and retrieval in data grids," *Future Generation Computer Systems*, vol. 23, pp. 536-546, May 2007.

[24] T. Hamrouni, S. Slimani, and F. B. Charrada, "A data mining correlated patterns-based periodic decentralized replication strategy for data grids," *Journal of Systems and Software*, vol. 110, pp. 10-27, Dec. 2015.

[25] U. Tos, R. Mokadem, A. Hameurlain, T. Ayav, and S. Bora, "Dynamic replication strategies in data grid systems: A survey," *Journal of Supercomputing*, vol. 71, pp. 4116-4140, Nov. 2015.

[26] R. Nou, S. Kounev, F. Julia, and J. Torres, "Autonomic QoS control in enterprise grid environments using online simulation," *Journal of Systems and Software*, vol. 82, pp. 486-502, Mar. 2009.

[27] R. Mokadem and A. Hameurlain, "Data replication strategies with performance objective in data grid systems: A survey," *International Journal of Grid and Utility Computing*, vol. 6, pp. 30-46, Dec. 2014.

[28] L. N. Nassif and J. M. Nogueira, "Resource selection in grid: A taxonomy and a new system based on decision theory, case-based reasoning, and fine-grain policies," *Concurrency and Computation: Practice and Experience*, vol. 21, pp. 337-355, Mar. 2009.

[29] Z. J. Li, C. T. Cheng, and F. X. Huang, "Utility-driven solution for optimal resource allocation in computational grid," *Computer Languages, Systems and Structures*, vol. 35, pp. 406-421, Dec. 2009.

[30] C. Curino, E. Jones, Y. Zhang, E. Wu, and S. Madden, "Relational cloud: The case for a database service," in *Proc. NEDS'10*, 2010, p. 1.

[31] S. Kunde and T. Mukherjee, "Workload characterization model for optimal resource allocation in cloud middleware," in *Proc. ACM SAC'15*, 2015, p. 442.

[32] K. A. Kumar, A. Quamar, A. Deshpande, and S. Khuller, "SWORD: Workload-aware data placement and replica selection for cloud data management systems," *VLDB Journal*, vol. 23, pp. 845-870, Dec. 2014.

[33] 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, Feb. 2010.

[34] A. Jahan, K. L. Edwards, and M. Bahraminasab, *Multi-Criteria Decision Analysis: For Supporting the Selection of Engineering Materials in Product Design*, 2nd ed., Oxford, UK: Butterworth-Heinemann, 2016.

[35] M. G. Rogers, M. Bruen, and L. Y. Maystre, *Electre and Decision Support: Methods and Applications in Engineering and Infrastructure Investment*, Heidelberg, Germany: Springer Science and Business Media, 2013.

[36] U. T. Mattsson. (2005) Database encryption-How to balance security with performance. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.127.8228&rep=rep1&type=pdf.

[37] Amazon Web Services Inc. (2010) Amazon CloudFront Pricing. [Online]. Available: http://aws.amazon.com/cloudfront/#pricing.

[38] N. Werstiuk. (2008) An enterprise perspective on energy efficiency and grids. [Online]. Available: http://www.ogf.org/OGF23/materials/1258/An+Enterprise+Perspective+on+Energy+Efficiency+and+Grids+OGF23.pdf

[39] T. Fan, J. Liu, and F. Gao, "Dynamic pricing strategy of shared devices in IIU federated cloud," *International Journal of Control and Automation*, vol. 9, pp. 199-219, 2016.

[40] V. Vusirikala, B. Koley, T. Hofmeister, V. Dangui, V. Kamalov, and X. Zhao, "Scalable and flexible transport networks for inter-datacenter connectivity," in *Proc. OFCC'15*, 2015, p. Tu3H-1.

[41] X. Zhang, A. Riska, and E. Riedel, "Characterization of the e-commerce storage subsystem workload," in *Proc. ICQES'08*, 2008, p. 297.

[42] K. Manassiev and C. Amza, "Scaling and continuous availability in database server clusters through multiversion replication," in *Proc. IEEE/IFIP ICDSN'07*, 2007, p. 666.

[43] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "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, Jun. 2001.

[44] I. M. Yassin, A. Zabidi, M. S. A. M. Ali, N. M. Tahir, H. A. Hassan, H. Z. Abidin, and Z. I. Rizman, "Binary particle swarm optimization structure selection of nonlinear autoregressive moving average with exogenous inputs (NARMAX) model of a flexible robot arm," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 6, pp. 630-637, Oct. 2016.

[45] M. N. M. Nor, R. Jailani, N. M. Tahir, I. M. Yassin, Z. I. Rizman, and R. Hidayat, "EMG signals analysis of BF and RF muscles in autism spectrum disorder (ASD) during walking," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 6, pp. 793-798, Oct. 2016.