Reputation-based Dependable Scheduling of Workflow Applications in Peer-to-Peer Grids

Mustafizur Rahman[a,*], Rajiv Ranjan[b], Rajkumar Buyya[a]

[a]*Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering*
The University of Melbourne, Victoria 3010, Australia
[b]*Service Oriented Computing (SOC) Research Group, School of Computer Science and Engineering*
The University of New South Wales, Sydney, Australia

Abstract

Grids facilitate creation of wide-area collaborative environment for sharing computing or storage resources and various applications. Inter-connecting distributed Grid sites through peer-to-peer routing and information dissemination structure (also known as Peer-to-Peer Grids) is essential to avoid the problems of scheduling efficiency bottleneck and single point of failure in the centralized or hierarchical scheduling approaches. On the other hand, uncertainty and unreliability are facts in distributed infrastructures such as Peer-to-Peer Grids, which are triggered by multiple factors including scale, dynamism, failures, and incomplete global knowledge.

In this paper, a reputation-based Grid workflow scheduling technique is proposed to counter the effect of inherent unreliability and temporal characteristics of computing resources in large scale, decentralized Peer-to-Peer Grid environments. The proposed approach builds upon structured peer-to-peer indexing and networking techniques to create a scalable wide-area overlay of Grid sites for supporting dependable scheduling of applications. The scheduling algorithm considers reliability of a Grid resource as a statistical property, which is globally computed in the decentralized Grid overlay based on dynamic feedbacks or reputation scores assigned by individual service consumers mediated via Grid resource brokers. The proposed algorithm dynamically adapts to changing resource conditions and offers significant performance gains as compared to traditional approaches in the event of unsuccessful job execution or resource failure. The results evaluated through an extensive trace driven simulation show that our scheduling technique can reduce the makespan up to 50% and successfully isolate the failure-prone resources from the system.

Keywords: Grid Computing, Workflow, Decentralized Management, Dependable scheduling, Peer-to-Peer Computing

1. Introduction

Grid computing enables the sharing, selection, and aggregation of geographically distributed heterogeneous resources, such as computational clusters, supercomputers, storage devices, and scientific instruments. These resources are under control of different Grid sites and being utilized to solve many important scientific, engineering, and business problems.

Inter-connecting distributed Grid sites through peer-to-peer routing and information dissemination structure (also known as Peer-to-Peer Grids) is essential to avoid the problems of scheduling efficiency bottleneck and single point of failure in the centralized or hierarchical scheduling approaches. Peer-to-Peer Grid (P2PG) model offers an opportunity for every site to pool its local resources as part of a single, massive scale resource sharing abstraction. P2PG infrastructures are large, heterogeneous, complex, uncertain and distributed.

In a P2PG, both control and decision making are decentralized by nature and different system components (users, services, application components) interact together to adaptively maintain and achieve a desired system wide behaviour. Furthermore, the availability, performance, and state of resources, applications and services undergo continuous changes during the life cycle of an application. Thus uncertainty and unreliability are facts in P2PG infrastructures, which are triggered by multiple factors, including: (i) software and hardware failures as the system and application scale that lead to severe performance degradation and critical information loss; (ii) dynamism (unexpected failure) that occurs due to temporal behaviours, which should be detected and resolved at runtime to cope with changing conditions; and (iii) lack of complete global knowledge that hampers efficient decision making as regards to composition and deployment of the application elements.

The aforementioned challenges are addressed in this paper by developing a novel self-managing [1] scheduling algorithm for workflow applications that takes into account the Grid site's prior performance and behaviour for facilitating opportunistic and context-aware placement of application components. The proposed scheduling algorithm is fully dependable, as it is capable of dynamically adapting to the changes in system behaviour by taking into consideration the performance metrics of Grid sites (software and hardware capability, availability, failure). The dependability of a Grid site is quantified using a decentralized reputation model, which computes local and global reputation scores for a Grid site based on the feedbacks provided by the scheduling services that have previously sub-

*Corresponding author. Tel.: +61 3 8344 1355; fax: +61 3 9348 1184.
Email addresses: mmrahman@csse.unimelb.edu.au (Mustafizur Rahman), rajiv@unsw.edu.au (Rajiv Ranjan), raj@csse.unimelb.edu.au (Rajkumar Buyya)

mitted their applications to that site. In particular, this paper **contributes** the following to the state-of-the-art in the Grid scheduling paradigm :

A novel Grid scheduling algorithm that aids the Grid schedulers such as resource brokers in achieving improved performance and automation through intelligent and opportunistic placement of application elements based on context awareness and dependability.

Further, the effectiveness of this contribution is appraised through:

(i) A comprehensive simulation-driven analysis of the proposed approach based on realistic and well-known application failure models to capture the transient behaviours that prevails in existing Grid-based e-Science application execution environments;

(ii) A comparative evaluation that demonstrates the self-adaptability of the proposed approach in comparison to Grid environments where: (1) resource/application behaviours do not change (i.e. no failure occurs), therefore no self-management is required and, (2) transient conditions exist but runtime systems and application elements have no capability to self-adapt.

The remainder of this paper is organized as follows. The related work that are focused on dependable application scheduling, distributed reputation models and Grid workflow management is presented in next section. Section 3 provides a brief discussion related to key system models in regard to overlay creation, application composition, task failure and application scheduling. In Section 4, we provide the distributed reputation management technique and the algorithms related to proposed dependable scheduling approach with example. Simulation setup, performance metrics and key findings of the experiments performed are analyzed and discussed in Section 5. Finally, we conclude the paper with the direction for future work.

2. Related Work

The main focus of this section is to compare the novelty of the proposed work with respect to existing approaches. We classify the related research into three main areas:

2.1. Dependable Scheduling

A recent work by Jik-Soo et al. [19] that advocates Content Addressable Network [28], DHT based dynamic propagation and load-balancing in desktop Grids, suffer from performance uncertainty and unreliability due to lack of context awareness in scheduling. A most recent proposal or reputation-driven scheduling in context of voluntary computing environments (desktop grids) has been put forward by Jason et al. [31]. They consider a centralized system model, where a central server is assigned responsibility for maintaining reliability ratings that form the basis for assigning tasks to group of voluntary nodes. Such centralized models for scheduling and reputation management [2] present serious bottleneck as regards to scalability of the system and autonomy of Grid sites. Moreover, these approaches are targeted on bag of tasks type of application model, whereas our approach considers scheduling of workflow applications. Currently, Grid information services [9], on which Grid schedulers [13] depend for resource selection, do not provide information regarding how the resources have performed in the recent past (performance history) or at what level they are rated by other schedulers in the system as regards to QoS satisfaction.

2.2. Distributed Reputation Models

There has been considerable amount of research work done in Peer-to-Peer (P2P) reputation systems to evaluate the trustworthiness of participating peers. These reputation systems are targeted towards P2P file sharing networks that focus on sharing and distribution of information in Internet-based environments. The PoweTrust model proposed by Zhou et al. [39], utilizes single dimensional Overlay Hashing Functions (OHFs) for: (i) aassigning score managers for peers in the system and (ii) aggregating/computing the global reputation score. These kinds of OHFs are adequate if the search for peers/resources is based on single keyword (such as file name) or where there is single ordering in search values. However, OHFs are unable to support (or support with massive overhead) searches containing multiple keywords, range queries (such as search for a Grid site that has: Linux operating system, 100 processors, Intel architecture, and reputation ≥ 0.5). The EigentTrust model [17] suggested by Kamvar et al. also suffers from the shortcomings mentioned above. To overcome these limitations, in the proposed approach a d-dimensional data distribution technique [26] is applied on the overlay of peers for managing the information related to complex searches and reputation values.

2.3. Grid Workflow Management

With the increasing interest in Grid workflows, many Grid workflow systems such as Pegasus [10], Triana [34], Taverna [22], Condor DAGMan [20], Kepler [21], SwinDeW-G [37], Gridbus [38] and Askalon [12] have been developed in recent years. Among these systems, in terms of workflow scheduling infrastructure, SwinDeW-G and Triana utilize decentralized P2P based technique. However, the P2P communication in SwinDeW-G and Triana is implemented by JXTA protocol, which uses a broadcast technique. In this work, we use a DHT (such as Chord) based P2P system for handling resource discovery and scheduling coordination. The employment of DHT gives the system the ability to perform deterministic discovery of resources and produce controllable number of messages in comparison to using JXTA.

3. System Models

3.1. Grid Model

The proposed scheduling algorithm utilizes the P2PG [27] model in regards to distributed resource organization and Grid networking.

Definition 1 (Peer-to-Peer Grid): The P2PG $G_p = \{S_1, S_2, ..., S_n\}$ *consists of a number of sites n; with each site contributing its local resource to the Grid. Every site in the P2PG has its own resource descriptor* D_i *which contains the definition of the resource that it is willing to contribute.* D_i

can include information about the CPU architecture, number of processors, memory size, secondary storage size, operating system type, etc.

In this work, $D_i = (p_i, a_i, s_i, o_i)$, which includes the number of processors p_i, processor architecture a_i, their speed s_i, and installed operating system type o_i. In Table 1, the definitions for symbols that are utilized in this paper are presented.

The application scheduling and resource discovery in the P2PG is facilitated by a specialized Grid Resource Management System (GRMS) known as Grid Autonomic Scheduler (GAS). Every contributing Grid site maintains its own GAS service. A GAS service is composed of the software components: Grid Autonomic Manager (GAM), Local Resource Management System (LRMS) and Grid Peer.

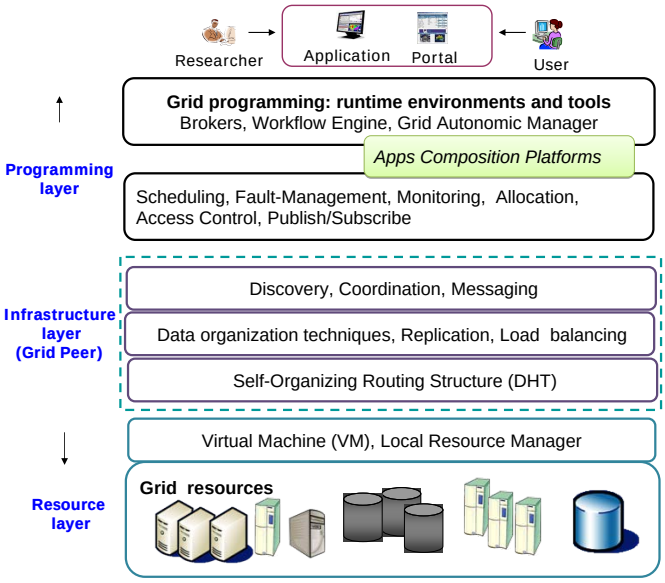


Figure 1: Layerd Design of Peer-to-Peer Grid architecture

The GAM component of GAS exports a Grid site to the outside world and is responsible for scheduling locally submitted jobs (workflows, parallel applications) in the P2PG. Further, it also manages the execution of remote jobs (workflows) in conjunction with the local resource management system. The LRMS software module can be realized using systems such as SGE (Sun Grid Engine) [15] and PBS [5]. Additionally, LRMS performs other activities for facilitating Grid wide job submission and migration process such as answering the GAM queries related to local job queue length, expected response time, and current resource utilization status.

P2PG requires supporting technologies to enable scalable collaboration and communication between resources and services across multiple Grid sites. For supporting the required functions, it is mandatory to build some kind of overlay network on top of the physical routing network. To this end, the Grid peer (see Fig. 1) implements an overlay (infrastructure level core services) for enabling decentralized and distributed resource discovery supporting resources status lookups and updates across the P2PG. It also enables decentralized inter-GAM collaboration for optimizing load-balancing and distributed resource provisioning. These core services are divided into a number of sub-layers (refer to Fig. 1): (i) higher level services for discovery, coordination, and messaging; (ii) low level distributed indexing and data organization techniques; and (iii) self-organizing overlay that builds over Distributed Hash Table (DHT) [32, 28] routing structure.

Table 1: Notations: Grid, Reputation, Failure models and Metrics

Symbol	Meaning
Grid	
n	number of sites or GASs in the Grid system
S_i	i-th Grid site in the system
GAS_i	i-th GAS in the system
a_i	processor architecture for resource at site S_i.
p_i	number of processors for resource at site S_i.
o_i	type of operating system for resource at site S_i.
s_i	processor speed for resource at site S_i.
Reputation	
$succ(i,j,k)$	output of result verification function for task T_k of S_j executed by S_i.
$feed(i,j,k)^t$	feedback score of task T_k from S_j for S_i after t transactions.
NF_i	total number of negetive feedbacks given by other sites for S_i.
TF_i^t	transaction feedback value for S_i after t transactions by S_i.
$TF_{i,j}^t$	transaction feedback value from S_j for S_i after t transactions.
GR_i^t	global reputation of S_i after t transactions.
$LR_{i,j}^t$	local reputation of S_i according to S_j after t transactions.
M_{LR}	local reputation matrix.
M_{GR}	global reputation matrix.
$LR_{initial}$	initial local reputation value of each site.
$GR_{initial}$	initial global reputation value of each site.
R_{th}	reputation threshold of a site for a task to be mapped by scheduler.
$\tau_{refresh}$	time interval after which initial value is assigned to reputation score of a site.
Failure	
fp_i	task failing probability of Grid site S_i.
X_Y	failure distribution, where X% sites fail task with probability between Y and $Y + 0.1$.
Metrics	
$M_{i,j,k}$	makespan of k-th workflow submitted by j-th user of i-th Grid site.
$M_{average}$	average makespan per workflow in the system.
F_i	number of tasks failed by site S_i.
F_{total}	total number of tasks failed in the system.
SCH_i	number of tasks scheduled by GAS_i.
SCH_{total}	total number of tasks scheduled in the system.
$\rho_{M_{average},F_{total}}$	Pearson's correlation coefficient between $M_{average}$ and F_{total}.

A Grid Peer service accepts three basic types of objects from the GAM service as regards to dependable and dynamic scheduling: (i) a *claim*, is an object sent by a GAM to the DHT overlay for locating the resources that match the user's application requirements, (ii) a *ticket*, is an update object sent by a Grid site, mentioning about the underlying resource conditions, and (iii) a *feedback*, is an object sent by a GAM regarding the reputation of a Grid site in the system upon the output arrival of a previously submitted task. Examples of *claim*, *ticket* and *feedback* objects are shown in Table 2, 3 and 4. In general, a

Table 2: Claims stored with the coordination service at time t

Time	Claim ID	$s_{x_{i,j,k}}$	$p_{x_{i,j,k}}$	$a_{x_{i,j,k}}$	$o_{x_{i,j,k}}$	Rank
200	Claim 1	> 800	1	Intel	Linux	0.2
350	Claim 2	> 1200	1	Intel	Linux	0.3
500	Claim 3	> 700	1	Sparc	Solaris	0.1
700	Claim 4	> 1500	1	Intel	Windows XP	0.4

Table 3: Ticket published to the coordiation service at time t

Time	GFA ID	s_i	p_i	p_i avail	a_i	o_i
900	GFA-8	1400	3	2	Intel	Linux

Grid resource is identified by more than one attribute (such as number of processors, type of operating system, CPU speed); so a claim, ticket or feedback object is always multi-dimensional. Further, each of these objects can specify different kinds of constraints on the attribute values. More details on how these objects are routed and mapped is given in Section 4.3.3.

3.2. Application Model

In this work, we consider the Scientific workflow applications as the case study for the proposed scheduling approach. A Scientific workflow application can modeled as a Directed Acyclic Graph (DAG), where the tasks in the workflow are represented as nodes in the graph and the dependencies among the tasks are represented as the directed arcs among the nodes. In general, a task in a workflow is a set of instructions that can be executed on a single processing element of a computing resource [7]. Examples of such workflow applications are [4], [35], [30], [11], and [24].

Definition 2 (Scientific Workflows): Scientific workflows describe a series of large number of structured activities and computations that arise in scientific problem solving. Usually, scientific workflows are data or computation intensive and the activities in the workflow have data or control dependencies among them.

Example 1: Let, V be the finite set of tasks $\{T_1, T_2, \ldots, T_x, \ldots, T_y, T_n\}$ of a workflow and E be the set of dependencies among the tasks of the form $\{T_x, T_y\}$ where, T_x is the parent task of T_y. Thus, the workflow can be represented as, $W = \{ V, E \}$.

In a workflow, an entry task does not have any parent task and an exit task does not have any child task. We also assume that a child task can not be executed until all of its parent tasks are completed. At any time of scheduling, the task that has all of its parent tasks finished, is called a ready task.

Table 4: Feedbacks sent by different Grid sites to the coordination service

Feedback ID	From	For	User ID	Workflow ID	Task ID	Score
002	S_3	S_1	1	1	4	1.0
040	S_2	S_9	1	2	6	0.5
100	S_2	S_9	1	2	9	0.42
251	S_5	S_{10}	1	3	89	0.5

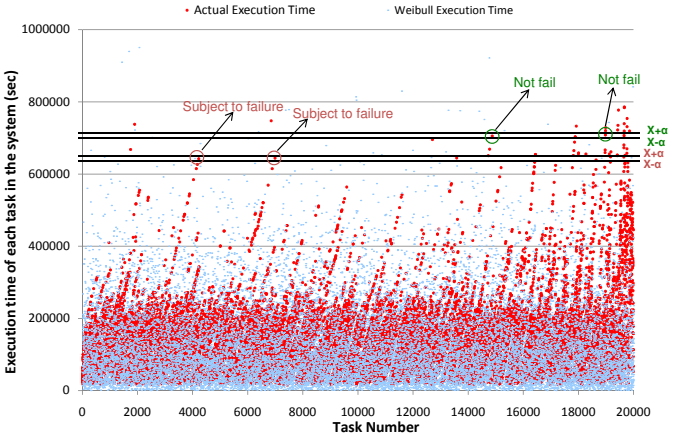


Figure 3: Determination of tasks likely-to-fail based on the distribution of experimental and Weibull task execution time.

3.3. Failure Model

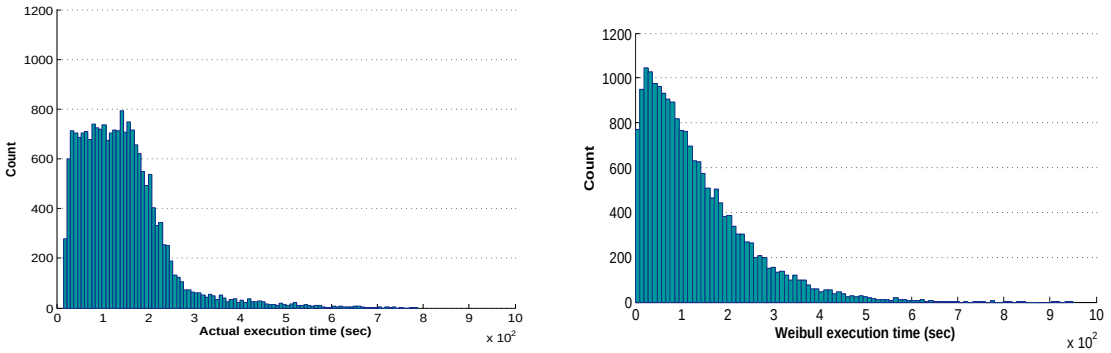
The Weibull distribution [18] is one of the most commonly used distributions in reliability engineering and has become a standard in reliability textbook for modeling time-dependant failure data. Therefore, in this work, we use a 2-parameter weibull distribution to determine whether a task execution is subject to failure or success in the system. The 2-parameter weibull distribution is generally characterized by two parameters: shape parameter, β and scale parameter, η. Fig. 2 shows the actual and weibull distribution of the task execution time in the system.

After a task has finished its execution on a resource, the execution time or the computational cost of that task is measured. If it falls within a certain range of the weibull distribution, then the task is considered as likely-to-fail. A task execution may fail for various reasons (e.g. the resource does not have appropriate libraries installed, executables are outdated or the resource has been restarted before sending all the output files). Thus, whether a task is likely to be failed is derived from weibull distribution and the logic for determining this is illustrated in Fig. 3.

Next, if a task is likely-to-fail, then whether it will be considered as a failed task further depends on the failure probability fp_i of the resource at site S_i that has been assigned to execute the task. For this, we generate a uniform random number between 0 and 1. If the value of this random number is less than fp_i, then the task is failed, otherwise it is successful.

Definition 3 (Failure Probability): Failure Probability, fp_i is defined as the likelihood or chance that the resource at Grid site S_i will fail the execution of a workflow task that is likely-to-fail in the system.

Example 2: Let us consider that failure probability of the resource at Grid site S_2 is 0.57. *Hence, S_2 will fail* 57 *of the* 100 *likely-to-fail tasks assigned to it for execution.*



(a) Histogram of task execution time based on experiments (Average execution time = 141 sec).

(b) Histogram of task execution time based on weibull distribution (β = 1.2, η = 141).

Figure 2: Distribution of task execution time.

4. Proposed Methodology

4.1. Distributed Reputation Management

In this section, we propose the key methods related to the distributed reputation management and its application to dependable scheduling.

In a fully decentralized and distributed Grid overlay, the P2P reputation system calculates the reputation score for a Grid site S_i by considering the opinions (i.e. feedbacks) [39, 17] from all the Grid sites $\in \{S_1, S_2, \ldots, S_n\}$, who have previously interacted with S_i. After a Grid site S_j completes a transaction with another Grid site S_i, S_j provides its feedback for S_i to the overlay, which is utilized to compute the reputation of S_i. This reputation value drives the future application scheduling decision making in choosing S_i for task execution. A Grid site, which accumulates higher reputation in the system is expected to be popular in the overlay. Over the period of time, the distributed scheduling services (GASs) in the system are more likely to prefer that site in the future for placement of tasks. On the other hand, a Grid site that performs badly over a period of time would accumulate comparatively lower reputation and will eventually be shunted out of the system, i.e. would receive none or very few job submissions from the schedulers (GAS).

In the proposed approach, the overlay maintains two reputation scores for each Grid site: (i) Global Reputation (GR) and (ii) Local Reputation (LR). Here, the GAS service (on behalf of local Grid site and users) rates the Grid sites, to which it submits a task, after every successful transaction (task completion) or unsuccessful transaction (task failure) based on a feedback function, $feed(i, j, k)$. The local and global reputation scores for Grid sites are stored within the distributed overlay in the form of local and global reputation matrix. These values are recursively aggregated from the feedback scores after each transaction and utilized by the scheduling algorithm to dynamically quantify the reliability of the sites.

4.1.1. Feedback Generation

GAS services can use a variety of rating functions based on system consensus for computing the feedback value. Some of the example functions can include the model used by *eBay* system. The reputation scheme in eBay is simple: +1 for a good or successful transaction, −1 for a poor or failed feedback, and 0 for a neutral or don't-care feedback. In this model, the feedback score has three discrete values, which evaluate the result of a transaction. However, this model does not incorporate different types of behaviour of the participating entities (e.g. an entity is failing transactions of only a particular entity, an entity is failing transactions only at the beginning or an entity is generating successful and unsuccessful transactions alternately) into the feedback score, which is required to be considered in case of heterogeneous and dynamic resource sharing Grid environments.

In our feedback model, the GAS service at site S_j computes the feedback, $feed(i, j, k)$ for a Grid site S_i dynamically after each transaction (i.e. S_i completes execution of a task T_k submitted by S_j). First, S_j verifies the output of a task returned by S_i using the result verification function $success(i, j, k)$ that assigns a value $\in \{0,1\}$, where 0 represents an unsuccessful/failed task execution and 1 represents a successful task execution. A task execution may fail for various reasons (e.g. the resource does not have appropriate libraries installed, executables are outdated or resource has been restarted before sending all the output files). The result verification function is represented as,

$$success(i, j, k) = \begin{cases} 1 & \text{if task execution is sucessful} \\ 0 & \text{if task execution is failed} \end{cases} \quad (1)$$

Then S_j generates the feedback score based on the value assigned by result verification function. If the assigned value is 1, feedback score is 1; on the other hand if the assigned valued is 0 then the feedback score is calculated from an exponential distribution. The output given by the exponential function (refer to Fig. 4) is varied over the number of failed transactions between the corresponding two Grid sites. The objective of using this exponential function is to give a Grid site greater opportunity to execute tasks at the beginning so that it is not shunted out of the system after only few failed transactions. However, if a site continues to fail more transactions, the value for exponential function approaches 0. Thus, if $F_{i,j}$ is the number of

unsuccessful task executions by S_i with S_j, the feedback score for task T_k, after t transactions by S_i with S_j can be represented as,

$$feed(i,j,k)^t = \begin{cases} 1 & \text{if } success(i,j,k) = 1 \\ \alpha_f^{\frac{1}{\beta_f}} & \text{if } success(i,j,k) = 0 \end{cases} \quad (2)$$

where, $0 < \alpha_f \leq 0.5$ and $\beta_f \in \{1, 2, 3\}$.

If the feedback score given by a Grid site S_j is 1, we consider it as Positive Feedback (PF), whereas a Negative Feedback (NF) is attained if feedback score is less than 1.

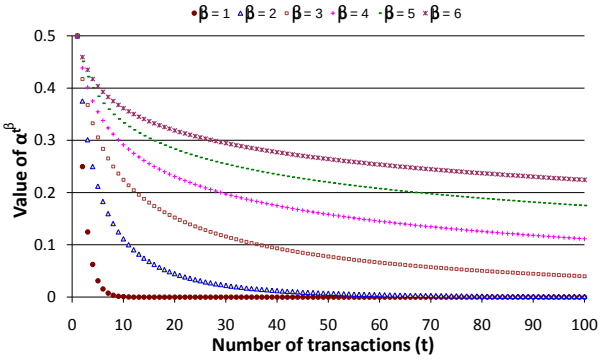


Figure 4: The growth of α^{β} over the number of transactions, t for different values of β. Here, $\alpha = 0.5$.

4.1.2. Global Reputation Calculation

The Global Reputation (GR) of a site is a statistical reputation that is calculated by averaging all the feedbacks given by the GAS services of other Grid sites for their tasks executed at that site. Once the overlay receives a feedback, it computes the Transaction Feedback (TF) for that feedback. The value of TF depends on whether the feedback is positive or negative. If negetive feedback is received, TF is same as the feedback value. However, if feedback is positive, the value of TF is computed from an exponential distribution (refer to Fig. 4), where the output value is varied over the total number of negative feedbacks received by the corresponding Grid site. The purpose of using this distribution is to allow a Grid site to accrue a higher value of GR only if it executes more successful tasks than failed tasks. So, if it fails very few transactions, the output of the exponential function reaches 1 accordingly. Thus, if NF_i is the total number of negative feedbacks given by other sites for S_i, the transaction feedback value after t transactions by S_i can be calculated as,

$$TF_i^t = \begin{cases} feed(i,j,k)^t & \text{if negative feedback or (positive feedback and } NF_i \text{ is 0)} \\ \{(1-\alpha_p)+\alpha_p^{NF_i^{\frac{1}{\beta_p}}}\} \times feed(i,j,k)^t & \text{if positive feedback and } NF_i > 0 \end{cases} \quad (3)$$

where, $0.5 \leq \alpha_p < 1.0$ and $\beta_p \in \{4, 5, 6\}$.

The GR of a particular Grid site is calculated by taking the average of aggregated TFs form other sites. Initially GR is assigned a value $GR_{initial}$ that is greater than or equal to the reputation threshold R_{th}. Afterwards, it is dynamically changed based on the TF computed after every transaction. Thus, GR of a Grid site, S_i after total t number of transactions with other sites is represented as,

$$GR_i^t = \begin{cases} GR_{initial} & \text{if } t = 0 \\ \frac{GR_i^{t-1} \times t + TF_i^t}{(t+1)} & \text{if } t > 0 \end{cases} \quad (4)$$

The GR value of each Grid site is stored in a matrix. At any instance of time, the DHT-based distributed overlay maintains $n \times 1$ global reputation matrix M_{GR} (refer to Fig. 5(a)) for all the Grid sites $S_i \in \{1,2,\ldots,n\}$ that is updated dynamically after every transaction in the system. This M_{GR} is utilized by the distributed scheduler for mapping tasks to the Grid sites based on their reputation values.

$$M_{GR} = \begin{pmatrix} 0.85 \\ 0.90 \\ 0.70 \end{pmatrix} \begin{matrix} S_1 \\ S_2 \\ S_3 \end{matrix} \qquad M_{LR} = \begin{matrix} \quad S_2 & S_2 & S_2 \end{matrix} \begin{pmatrix} 0.95 & 0.82 & 0.75 \\ 0.75 & 0.98 & 0.82 \\ 0.88 & 0.56 & 0.76 \end{pmatrix} \begin{matrix} S_1 \\ S_2 \\ S_3 \end{matrix}$$

(a) Global reputation matrix

(b) Local reputation matrix

Figure 5: Reputation matrix for three Grid sites (S_1, S_2, S_3).

4.1.3. Local Reputation Calculation

Sometime, considering only GR of a Grid site for mapping tasks, cannot guarantee dependable scheduling. For example, the resource at a site S_i may fail tasks submitted by only a particular Grid site S_j. In this case, as S_j successfully executes tasks submitted by other Grid sites, its GR is high. So, the scheduler may still map the tasks submitted by S_j to S_i. Therefore, we introduce another reputation score, Local Reputation (LR) for a Grid site.

Similar to GR, LR is calculated as an average of the feedback values except it considers feedbcks from only one Grid site. TF for computing LR also follows the same function as generating TF for GR. Therefore, if $NF_{i,j}$ is the number of negative feedbacks given by S_j for S_i after t transactions with S_i, the transaction feedback value can be calculated as,

$$TF_{i,j}^t = \begin{cases} feed(i,j,k)^t & \text{if negative feedback or (positive feedback and } NF_i \text{ is 0)} \\ \{(1-\alpha_p)+\alpha_p^{NF_{i,j}^{\frac{1}{\beta_p}}}\} \times feed(i,j,k)^t & \text{if positive feedback and } NF_i > 0 \end{cases} \quad (5)$$

where, $0.5 \leq \alpha_p < 1.0$ and $\beta_p \in \{4, 5, 6\}$.

Now, the LR of a Grid site, S_i according to S_j, after t number of transactions with S_j is represented as,

$$LR_{i,j}^t = \begin{cases} LR_{initial} & \text{if } t = 0 \\ \frac{LR_{i,j}^{t-1} \times t + TF_{i,j}^t}{(t+1)} & \text{if } t > 0 \end{cases} \quad (6)$$

The LR values of each Grid site in regards to other sites are kept in a $n \times n$ local reputation matrix M_{LR} (refer to Fig. 5(b)), which is stored in the overlay and updated dynamically after every transaction between the corresponding sites. Similar to M_{GR}, M_{LR} is also utilized by the distributed scheduler for mapping tasks to the Grid sites based on their reputation values.

An example scenario of local and global reputation calculation in the distributed coordination space is depicted in Fig. **??**.

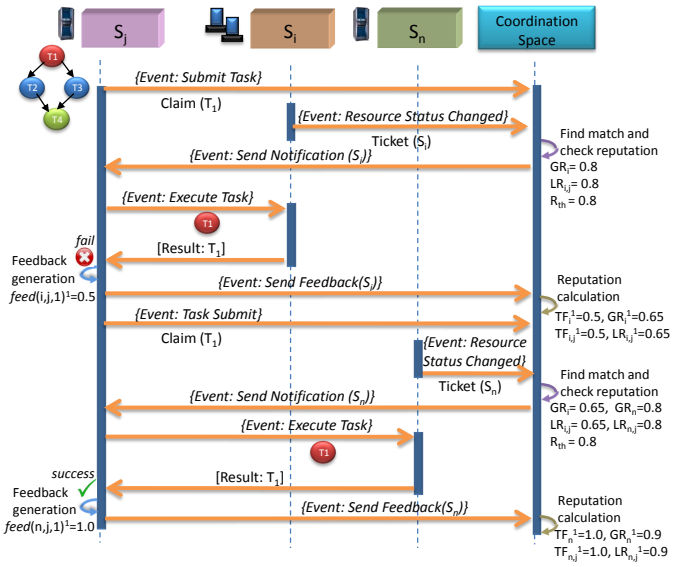


Figure 6: Interaction among different Grid entities in reputation based dependable workflow scheduling approach.

Algorithm 1 TASK SCHEDULING AT GAS

1: **PROCEDURE**: Event- User Workflow Submit
2: Input: Workflow W
3: **begin**
4: Calculate rank value for each task using EFT heuristic
5: Generate *Ready TaskList* for W
6: Submit *Ready* tasks for execution
7: **end**
8: **PROCEDURE**: Event- Task Finish Notification
9: Input: Task T_k, Workflow W
10: **begin**
11: Update dependency list of each task in $TaskList$
12: Generate *Ready TaskList* for W
13: Submit *Ready* tasks for execution
14: **end**
15: **PROCEDURE**: Generate Feedback
16: Input: Task T_k, Site S_i
17: **begin**
18: Verify output of T_k by (1)
19: **if** $success(i, j, k) = 0$ **then**
20: $F_{i,j} \leftarrow F_{i,j} + 1$
21: **end if**
22: Calculate feedback score for T_k by (2)
23: **end**

4.2. Distributed Workflow Management

In this section, we provide the description of the algorithms that have been devised for task scheduling and resource provisioning in order to achieve reputation-based workflow management.

4.2.1. Task Scheduling

Here, we discuss about the task scheduling algorithm (refer to Algorithm 1) that is undertaken by a GAS in P2PG on arrival of a job or workflow. When a user submits a workflow application W, the GAS calculates the priority of each task (line 4). Earliest Finish Time (EFT) [36] heuristic is used to calculate task priorities by traversing the task graph in Breadth First Search (BFS) manner. Once the rank values are calculated, the GAS generates *Ready* tasks in the *TaskList* based on the dependency of each task and put them into the *Ready TaskList* (line 5). Finally, GAS submits the *Ready* tasks for execution (line 6).

Further, when the GAS receives a notification message from site S_i stating task T_k has finished execution, it first updates the dependency lists of the tasks that are dependant on T_k (line 11); then it computes the *Ready* tasks at that moment and submits those for execution (line 12-13). Next, it generates feedback for the transaction with S_i (line 15). In order to do that it first verifies the output of T_k using the result verification given by equation (1) (line 18). If output of the function is 0, $F_{i,j}$ is incremented by one (line 20). Then it calculates the feedback score for this transaction by equation (2) (line 22).

4.2.2. Resource Provisioning

The details of the decentralized resource provisioning algorithm (refer to Algorithm 2) that is undertaken by the P2P coordination space is presented here. When a resource claim object r_k arrives at the coordination service, it is added to the existing claim list, *ClaimList* by the coordination service (line 1-5). When a resource ticket object u_i arrives at coordination service, the list of resource claims ($ClaimList_m$) that overlap or match with the submitted resource ticket object is computed (line 6-10) if global reputation of that resource is greater than or equal to the reputation threshold R_{th}. The overlap signifies that the task associated with the given claim object can be executed on the ticket issuer's resource subject to its availability.

Then the coordination service sorts the claim objects in $ClaimList_m$ in descending order according to their rank value (line 11). From the $ClaimList_m$, the resource claimers are selected one by one based on their rank value(higher rank first) and notified about the resource ticket match if local reputation of ticket issuer against the resource claimer is greater than or equal to R_{th} and until the ticket issuer is not over-provisioned (line 13-19).

When a feedback object is arrived at coordination service, first it is decided whether the feedback is negetive or positive. If feedback is negetive, NF_i and $NF_{i,j}$ are incremented by one (line 25-27). Then local and global reputation scores are calculated consequently (line 29-30). Finally, the local and global reputation matrices that are stored in coordination space are updated by the coordination service (line 31).

Algorithm 2 RESOURCE PROVISIONING AT COORDINATION SPACE

1: **PROCEDURE**: Event- Claim Submit
2: Input: Claim r_k
3: **begin**
4: $ClaimList \leftarrow ClaimList \cup r_k$
5: **end**
6: **PROCEDURE**: Event- Ticket Submit
7: Input: Ticket u_i from Resource R_i
8: **begin**
9: **if** $GR_i \geq R_{th}$ **then**
10: $ClaimList_m \leftarrow$ list of claims in $ClaimList$ that are matched with u_i
11: Sort $ClaimList_m$ in descending order of task's rank
12: $index \leftarrow 0$
13: **while** R_i is not over-provisioned **do**
14: **if** $LR_{i,j} \geq R_{th}$ **then**
15: Send notification of match event to resource claimer $ClaimList_m$[index]
16: Remove $ClaimList_m$[index]
17: $index \leftarrow index + 1$
18: **end if**
19: **end**
20: **end if**
21: **end**
22: **PROCEDURE**: Event- Feedback submit
23: Input: Feedback from site S_j
24: **begin**
25: **if** feedback is negetive **then**
26: $NF_i \leftarrow NF_i + 1$
27: $NF_{i,j} \leftarrow NF_{i,j} + 1$
28: **end if**
29: Calculate TF_i and $TF_{i,j}$ by (3) and (5)
30: Calculate GR_i and $LR_{i,j}$ by (**??**) and (**??**)
31: Update M_{GR} and M_{LR}
32: **end**

4.2.3. Time Complexity

This section analyses the computational tractability of the approach by deriving several time complexity bounds to measure the computational quality. Using the example for Grid workflow application model, we analyse the complexity of calculating tasks ranks, feedback generation, and reputation scores (see Algorithm 1). These complexities are further aggregated to model a composite function that represents the overall complexity.

We consider a P2PG infrastructure consisting of n number of Grid sites. Every Grid site S_i instantiates a service GAS_i. This implies that there are total n number of GAS services in the infrastructure that are continuously injecting task to resource mapping requests in form of Claim Objects (refer to line 1 in Algorithm 2). We assume every user submits a workflow application consisting of T number of tasks and E number of dependencies among the tasks to its local GAS service. Then the complexity of calculating rank values of all the tasks using EFT heuristic through BFS is $O(E + T)$ [8]. Further, if an adjacency list is used to handle the dependencies, then the complexity of generating *Ready* tasks and updating dependency list is $O(E)$ [8].

Next, we derive the time complexity of generating feedback in Algorithm 1 (lines 15-23). After a GAS service receives the output for the submitted task, it has to compute a feedback score, which has to be reported to peer-to-peer overlay. The feedback score calculation involves few mathematical computation (see Eq. 2), therefore it involves constant complexity of $O(1)$. Therefore, the overall time complexity of Algorithm 1 is $O(E + T)$.

In worst case, *ClaimList* in the Algorithm 2 can contain $n.T$ number of enteries. So the complexity of sorting the *ClaimList* is $O((n.T)\log(n.T))$ (through the implementation of merge sort algorithm) and finding out the total number of matches is $O(n.T)$. Calculating the number of resource claimers that has to be notified about the matches also requires $O(n.T)$ steps in worst case.

Every new feedback score submitted by GAS services has to be aggregated into global reputation score (using Eq. 4). Similar to feedback computation, updating global reputation score also involves series of mathematical steps. Hence, the overall complexity of computing or updating global reputation score is constant, $O(1)$.

Finally, the adjacency matrix also handles the reputation matrices. Here, updating M_{GR} and M_{LR} is also bounded by $O(1)$. Thus the overall complexity of Resource Provisioning algorithm is $O((n.T)\log(n.T))$.

4.3. Distributed Overlay Management

In order to create a collaborative environment and achieve efficient and scalable resource lookup, a peer-to-peer Grid overlay is created and utilized in the proposed approach. A Grid peer undertakes the following critical tasks related to management of this overlay, which are important for proper functioning of P2PG:

4.3.1. Overlay Construction

The overlay construction refers to how Grid peers are logically connected over the physical network. In this work, we utilize Chord [32] as the basis for creation of Grid peer overlay. A Chord overlay inter-connects the Grid peer services based on a ring topology. Fig. 7 shows a Chord-based Grid peer overlay. The objects and Grid peers are mapped on the overlay depending on their key values. Each Grid peer is assigned responsibility for managing a small number of objects and building up routing information (finger table) at various Grid peers in the network. In Fig. 7 Grid peers including 2, 8, and 14 have a finger table of size 4. The finger table aids in resolving the lookup request within acceptable bounds such as in $O(\log(n))$ routing hops. The finger table is constructed when a Grid peer joins the overlay, and it is periodically updated to take into account any new joins, leaves or failures.

4.3.2. Multi-dimensional Data Indexing

Traditionally, Chord as well as other DHT overlays, such as CAN [28], Pastry [29]) have been proved to be efficient for indexing 1-dimensional data (e.g. find a Grid resource that offers "Pentium" processor). However, resources hosted by a Grid site are identified by more than one attribute; thereby a claim or a ticket or a feedback object is always multi-dimensional in nature. In order to support multi-dimensional data indexing (processor type, OS type, CPU speed) over Chord overlay, we have implemented a spatial indexing technique [33].

The indexing technique builds a multi-dimensional attribute space based on the Grid resource attributes, where each attribute represents a single dimension. An example 2-dimensional attribute space that indexes resource attributes including Speed and CPU Type is shown in Fig. 7.

The attribute space resembles a grid like structure consisting of multiple index cells. Each index cell is uniquely identified by its centroid, termed as the *control point*. The Chord hashing method (DHash(coordinates)) is used to map these control points so that the responsibility for an index cell is associated with a Grid peer in the overlay. For example in Fig. 7, DHash($x1$, $y1$) = $k10$ is the location of the control point A ($x1$,$y1$) on the overlay, which is managed by Grid peer 12.

4.3.3. Object Mapping and Routing

This process involves identification of index cells in the attribute space to map a claim, ticket, or a feedback object. For mapping claims, a mapping strategy based on diagonal hyperplane of the attribute space is utilised. This mapping involves feeding candidate claim index cells as inputs into a mapping function, $Imap(claim)$. This function returns the IDs of index cells to which the given claim can be mapped (refer to step 7 in Fig. 7). Distributed hashing ($DHash(cells)$) is performed on these IDs, which returns keys for Chord overlay to identify the current Grid peers responsible for managing the given keys. Similarly, mapping of ticket and feedback objects also involves the identification of the cell in the attribute space using the same algorithm.

4.4. Scheduling Example

This section provides an example scenario of the process of task scheduling and distributed reputation management. The key steps involved with the proposed scheduling approach (see Fig. 8) are as follows:

1. A user submits his task to the local GAS service at site S_u.
2. Following this, the GAS inserts a claim object to the DHT-based overlay to locate a *dependable* and *available* Grid site (resource) that has reasonable reputation rating (above reputation threshold) in the system.
3. The GAS, GAS_s at site S_s submits a ticket object to the overlay encapsulating the information about status (availability) of the local resource.
4. The overlay undertakes the decentralized matchmaking mechanism and discovers that the resource ticket issued by Grid site S_s matches with the resource description and reputation rating currently specified by claim object inserted by site S_u. Following that a match notification message is sent to S_u.
5. Next, GAS_u sends the task to site S_s. While the application is being processed, GAS_u periodically monitors the execution progress by sending *IsAlive* messages to S_s. IsAlive messages allow the GAS services to detect the hardware and network link failure related to the site S_s.
6. Once the execution of the task is finished, S_s returns the output to GAS_u.
7. Finally, GAS_u performs the result verification for the received output, computes the feedback score for S_s and reports to the overlay. The feedback score is aggregated to the local and global reputation scores for S_s using the proposed decentralized and distributed reputation model, described in the next section.

5. Performance Evaluation

5.1. Simulation Setup

Our simulation infrastructure is created by combining two discrete event simulators namely *GridSim* [6], and *PlanetSim* [14]. GridSim offers a concrete base framework for simulation of different kinds of heterogeneous resources, services and application types. PlanetSim is an event-based overlay network simulator that can simulate both unstructured and structured overlays.

5.1.1. Workload Configuration

In this study, we consider fork-join workflow (see Fig. 9) and an example of such workflow is WIEN2K [4], which is a quantum chemistry application developed at Vienna University of Technology. In this kind of workflow, forks of tasks are created and then joined, such that there can be only one entry task and one exit task. We vary the number of tasks in a workflow from 100 to 500 during the experiments and the size of each task is randomly generated from a uniform distribution between 50000 MI (Million Instructions) to 500000 MI. Further, we assume that workflows are computation intensive. Thus, the data

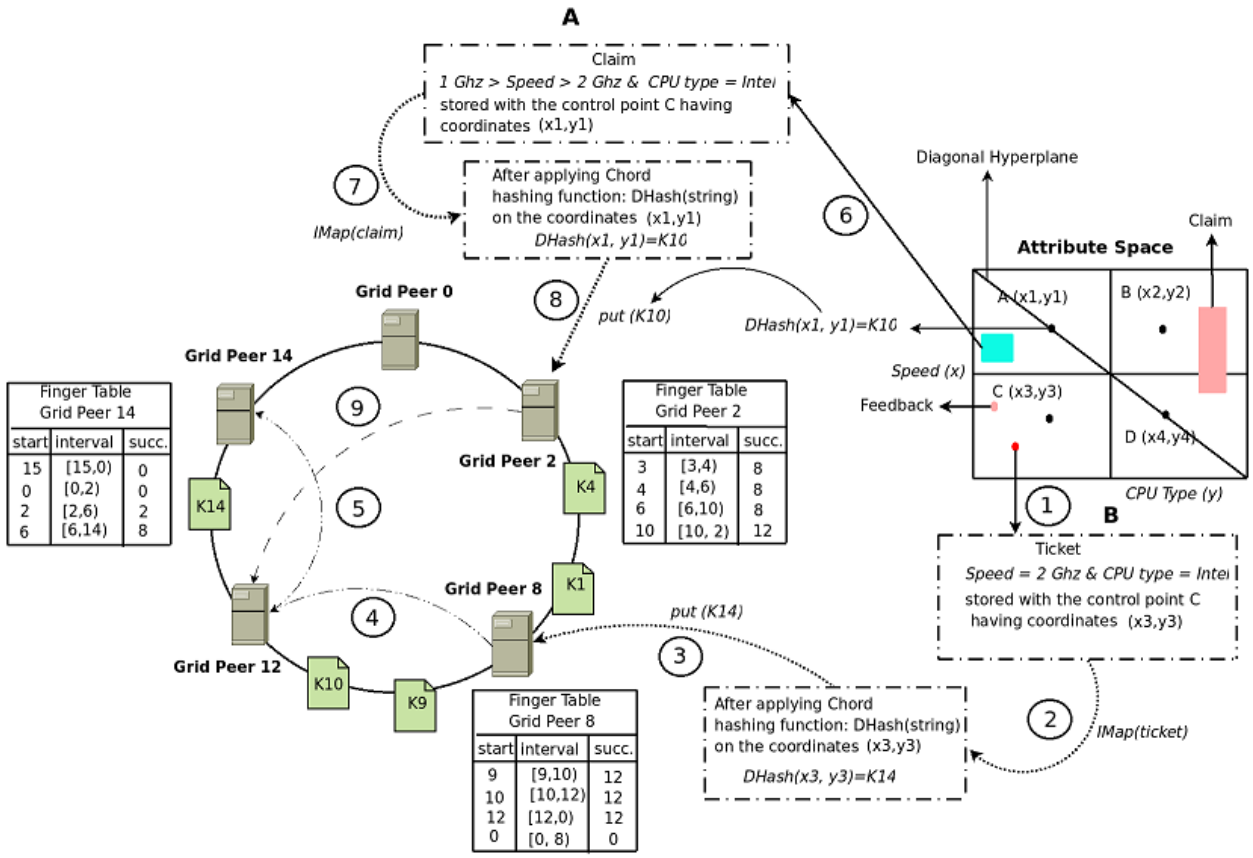


Figure 7: Overlay creation, data indexing, object mapping and routing: (1) a Grid site publishes ticket; (2) Grid peer 8 service computes the index cell, C($x3$,$y3$), to which the ticket maps by using mapping function $IMap$(ticket); (3) Next, distributed hashing function, DHash($x3$, $y3$), is applied on the cell's coordinate values, which yields a overlay key, $K14$; (4) Grid peer 8 based on its finger table entry forwards the request to peer 12; (5) Similarly, peer 12 on the overlay forwards the request to peer 14; (6) a GAS service submits a resource claim; (7) Grid peer 2 computes the index cell, C($x1$, $y1$), to which the claim maps; (8) DHash($x1$, $y1$) is applied that yields an overlay key, $K10$; (9) Grid peer 2 based on its finger table entry forwards the mapping request to peer 12.

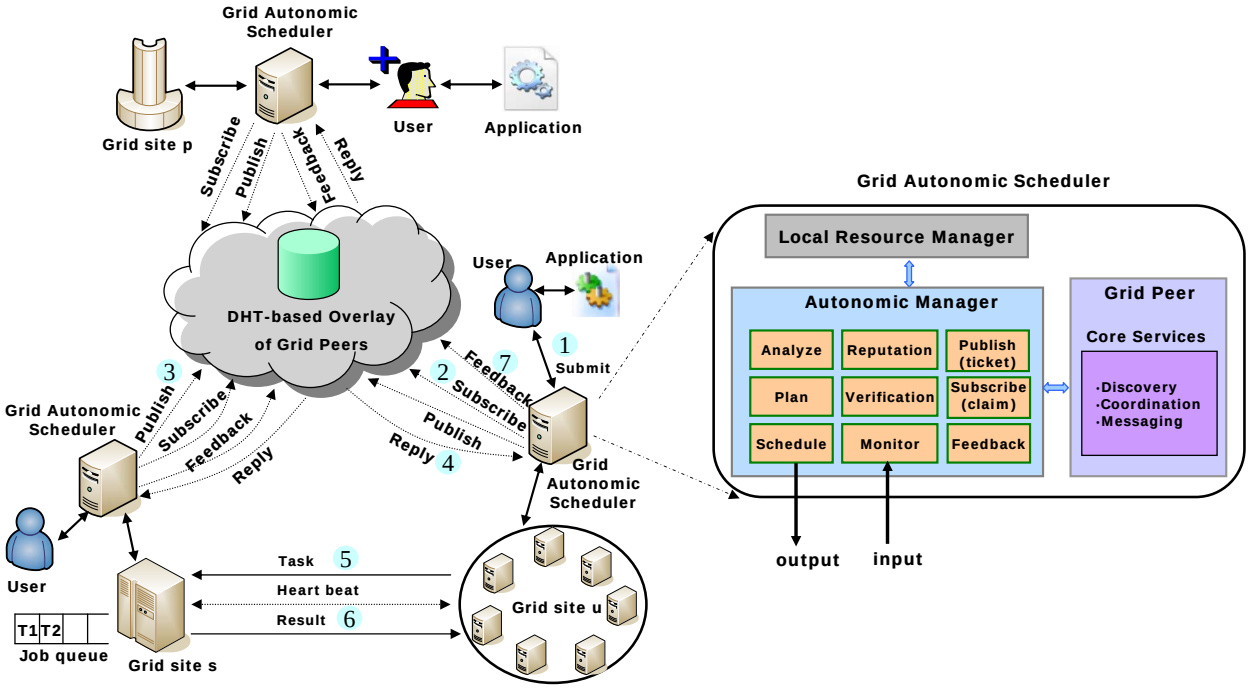


Figure 8: Reputation-based dependable scheduling example. Grid sites p, l, s, and u are managed by their respective Grid Autonomic Scheduler services.

dependency among the tasks in the workflow is negligible. In the Grid federation, each site has one user and each submits one workflow for execution.

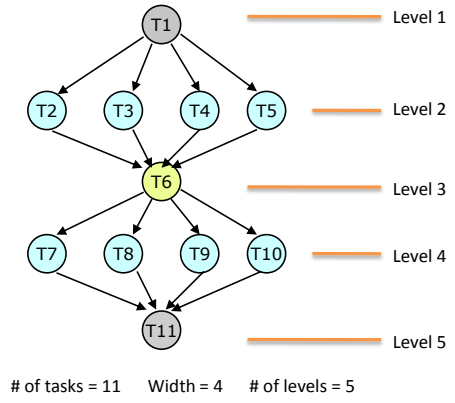


Figure 9: A fork-join workflow.

5.1.2. Network Configuration

The experiments run a Chord overlay with 32 bit configuration (number of bits utilized to generate node and key ids). The total number of GAS/broker in the system is 64. Further, network queue message processing rate is fixed at 4000 messages per second and message queue size is fixed at 10^4.

5.1.3. Resource Claim and Ticket Injection Rate

The GASs inject the ticket objects based on the exponential inter-arrival time distribution. The injection rate (i.e. resource update query rate) for the resource tickets is every 200 seconds [23]. At the beginning of the simulation, the resource claims for the entry tasks of all the workflows in the system are injected. Subsequently, when these tasks finish, then the resource claims for the successive tasks in the workflow are posted. This process is repeated until all the tasks in the workflow are successfully completed. Spatial extent of both resource claims and ticket objects lie in a 4-dimensional attribute space (an example is shown in Fig. 10). These attribute dimensions include the number of processors, p_i, their speed, s_i, their architecture, a_i, and operating system type, o_i. The distribution for these resource dimensions is generated by utilizing the configuration of resources that are deployed in various Grids including NorduGrid, AuverGrid, Grid5000, NaregiGrid, and SHARCNET [16].

5.1.4. Reputation Configuration

The values of the parameters for configuring reputation based scheduling in our experiment are listed in Table 5.

Table 5: Reputation parameters

Parameter	Value	Parameter	Value
α_f	0.5	$LR_{initial}$	0.8
β_f	2.0	$GR_{initial}$	0.8
α_p	0.5	R_{th}	0.8
β_p	5.0	$\tau_{refresh}$	1000 sec

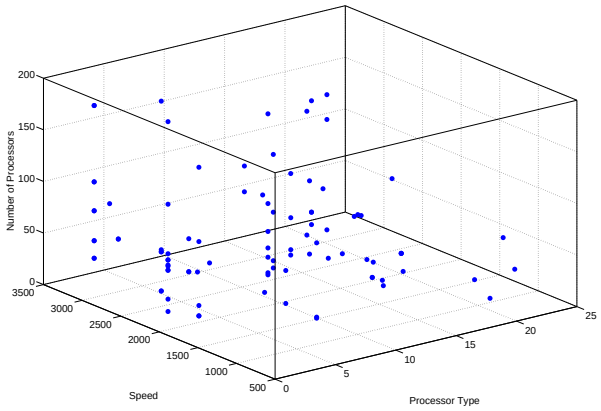


Figure 10: 3-dimensional attribute space for resource configuration and ticket data distribution.

5.1.5. Failure Configuration

In our experiment, the values of weibull shape and scale parameters β and η are 1.2 and 141 respectively, where the mean execution time of a task in the system is equal to 141 sec.

Along with this Weibull distribution, we also generate a set of resource failure distributions, X_Y by incorporating resource failure probability fp, where X represents the percentage of resources likely to fail tasks in the system and Y represents the probability of failure. For instance, if X is 20 and Y is 0.4, then 20% of resources in the system may fail tasks with the probability (fp) between 0.4 and 0.5. The resource failure distributions, we use in the experiment are as follows:

$\boldsymbol{X\_0.1}$: $\mathbf{0.1 \leq fp < 0.2}$; $\boldsymbol{X\_0.3}$: $\mathbf{0.3 \leq fp < 0.4}$

$\boldsymbol{X\_0.5}$: $\mathbf{0.5 \leq fp < 0.6}$; $\boldsymbol{X\_0.7}$: $\mathbf{0.7 \leq fp < 0.8}$

$\boldsymbol{X\_0.9}$: $\mathbf{0.9 \leq fp < 1.0}$

Some example failure distributions are presented in Fig. 6 and Fig. 7.

Table 6: Example failure distributions (25_Y)

ResourceID	25_0.1	25_0.3	25_0.5	25_0.7	25_0.9
1	0	0	0	0	0
2	0	0	0	0	0
3	0.1542	0.3390	0.5013	0.7864	0.9662
4	0	0	0	0	0

Table 7: Example failure distributions (50_Y)

ResourceID	50_0.1	50_0.3	50_0.5	50_0.7	50_0.9
1	0	0	0	0	0
2	0.1787	0.3655	0.5352	0.7573	0.9614
3	0.1135	0.3719	0.5884	0.7117	0.9418
4	0	0	0	0	0

5.2. Performance Metrics

As a measurement of scheduling performance, we evaluate the following performance metrics:

Scheduling Efficiency: In order to determine the scheduling efficiency, we measure two values of the system: (i) *average makespan per workflow* and (ii) *total number of tasks failed* by all the Grid sites in the system.

Definition 4 (Makespan): Makespan is calculated as the response time of a whole workflow, which is equal to the difference between the submission time of the entry task in the workflow and the output arrival time of the exit task in that workflow.

Example 3: Let us consider that a user at Grid site $\boldsymbol{S_i}$ wants to execute a fork-join workflow illustrated in Fig. 9, consisting of $\mathbf{11}$ *tasks. If $\boldsymbol{GAS_i}$ submits a claim object for task $\boldsymbol{T_1}$ to the overlay at time $\boldsymbol{t_1 = 20}$ sec and the output of task $\boldsymbol{T_{11}}$ is delivered to the user at time $\boldsymbol{t_2 = 1220}$ sec, then the makespan of this workflow is $\boldsymbol{t_2 - t_1 = 1200}$ sec.*

The measurement of makespan is taken by averaging over all the workflows in the system. If there are $\boldsymbol{n}$ number of Grid sites and each site has $\boldsymbol{u}$ number of users with each user submitting $\boldsymbol{w}$ number of workflows, then average makespan per workflow in the system can be defined as,

$$M_{average} = \frac{\sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq u \\ 1 \leq k \leq w}} M_{i,j,k}}{n \times u \times w}$$

If there are $\boldsymbol{n}$ number of Grid sites and site $\boldsymbol{S_i}$ fails $\boldsymbol{F_i}$ number of tasks, then the *total number of tasks failed* in the system can be defined as,

$$F_{total} = \sum_{1 \leq i \leq n} F_i$$

Scheduling complexity: It is measured as the *total number of task scheduled* by all GASs in the system. If there are $\boldsymbol{n}$ number of Grid sites and $\boldsymbol{GAS_i}$ schedules $\boldsymbol{SCH_i}$ number of tasks, then *total number of task scheduled* in the system can be expressed as,

$$SCH_{total} = \sum_{1 \leq i \leq n} SCH_i$$

Pruning Efficiency: We consider pruning efficiency as the degree to which the failure-prone resource are shunted out of the system. We have measured *total number of tasks successfully executed and failed* by the resource at each Grid site in order to show the pruning efficiency.

5.3. Results and Observations

In this section, we present the experimental results obtained by simulating our reputation based dependable workflow scheduling approach and compare these with that of other approaches. The experiments are conducted with the aim at characterizing:

(i) the performance of proposed reputation based dependable scheduling approach (*Failure with Reputation*), compared to its alternatives, *No Failure* (resources do not fail any task) and *Failure without Self-adaptation* (some resources fail tasks and

scheduler uses a simple rescheduling technique) with respect to various performance metrics;

(ii) the impact of different resource failure distributions and sizes of workflow on the performance of our approach and of its alternatives;

(iii) the significance of reputation threshold (R_{th}) on the performance of proposed reputation based scheduling approach.

(iv) the performance of the exponential feedback function utilized in the proposed reputation based scheduling approach.

The configuration of different parameters for all the experiments are listed in Fig. 8.

5.3.1. Experiment 1: Measuring Scheduling Efficiency

Experiment 1.1 (Impact of failure distribution): Fig. 5.3 presents the results of scheduling efficiency of the proposed reputation based scheduling approach against the other approaches, *Failure without Self-adaptation* and *No Failure*. The total number of tasks failed by all Grid sites, F_{total} for each of the three approaches are depicted in Fig. 11(a) and Fig. 11(b) for different failure distributions. As we can see from Fig. 11(a) that when the failure probability of the resources is increased (for example, from **0.1** to **0.9**), F_{total} in *Failure without Self-adaptation* is heavily increased accordingly.

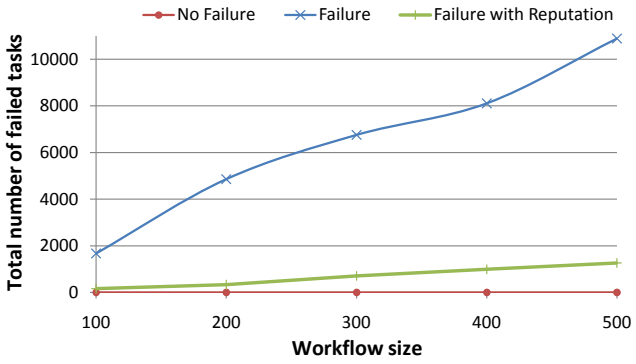
This situation is further aggravated for **50_*Y*** (refer to Fig. 11(b)) since more resources are likely to fail tasks. In contrast, our approach, *Failure with Reputation* can strongly reduce the number of task failures in the system irrespective of failure distributions. This happens due to the reason that in this case, the resources with higher failure probability are not assigned any task by the schedulers as their reputation scores are decreased beyond the threshold R_{th} after few task failures. Therefore, F_{total} in *Failure with Reputation* is not increased with the increase in failure probability since those failure-prone resources are always shunted out of the system after few failures. For instance, total number of tasks failed by all sites in *Failure with Reputation* is upto **96.8%** and **96.5%** less than that in *Failure without Self-adaptation* for **25_*Y*** and **50_*Y*** respectively.

The average makespan per workflow, $M_{average}$ also shows (see Fig. 11(c) and Fig. 11(d)) similar trend (upto **28%** and **50%** makespan reduction for **25_*Y*** and **50_*Y*** respectively) as reflected in total number of task failures since if one task is failed, its child tasks can not be scheduled and eventually the completion time of the whole workflow is increased.

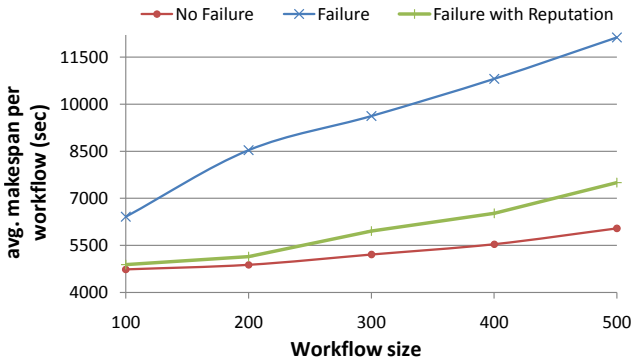
Experiment 1.2 (Impact of number of tasks in workflow): Fig. 12 presents the results of scheduling efficiency of the proposed reputation based dependable scheduling approach against the other approaches, *Failure without Self-adaptation* and *Failure* for different sizes of workflow. The results show that if the number of tasks in a workflow increases, $M_{average}$ also increases for all the three approaches since the overall workload on the system is increased. But the impact is more evident for *Failure without Self-adaptation*.

As we can see from Fig.12(a) and Fig.12(b), in case of *Failure without Self-adaptation*, both F_{total} and $M_{average}$ are increased rapidly with the increase in workflow size (number of tasks). This happens due to the reason that when the workflow size is increased, average number of tasks scheduled per resource in the system is also increased linearly as the number of Grid sites is not changed over time in this experiment. This results in allowing the failure-prone resources to fail more tasks. Thus, F_{total} and $M_{average}$ in *Failure without Self-adaptation* show a piecewise linear growth over the size of workflow.

However, in case of *Failure with Reputation*, when the workload on the system is increased, resources with higher reputation score get more tasks leaving the failure-prone resources isolate. This results in less number of task failures in the system even in higher workload. Therefore, with the increase in workflow size across the system, the performance gain achieved in terms of F_{total} and $M_{average}$ by applying the proposed approach is more evident. For example, when the workflow consists of 500 tasks, F_{total} in *Failure with Reputation* is **88.4%** less than that in *Failure without Self-adaptation* and the makespan reduction is **38.1%** accordingly.



(a) Total task failures vs. Number of tasks in workflow

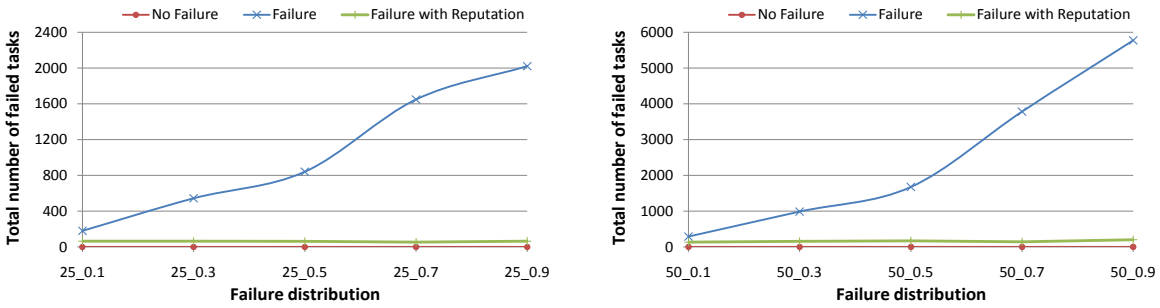


(b) Makespan vs. Number of tasks in workflow

Figure 12: Effect of workflow size on F_{total} and $M_{average}$ in the system (failure distribution **50_0.5**).

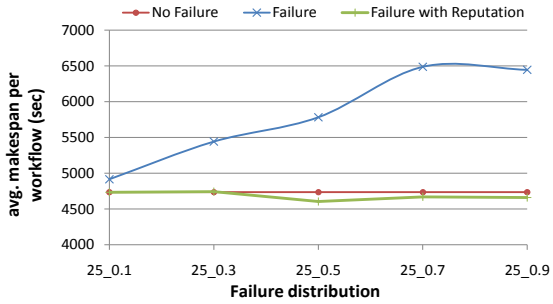
5.3.2. Experiment 2: Measuring Scheduling Complexity

Fig. 13 shows the total number of tasks scheduled by GAS1 to GAS16 in the system for the failure distribution, **50_0.5**. From the figure, it is evident that in case of *Failure without Self-adaptation*, each GAS needs to schedule more tasks than *No Failure* (where, GAS is not required to schedule any extra task than the size of workflow), which increases the load on

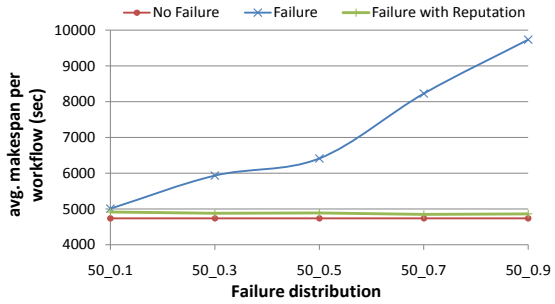


(a) Total task failures for failure distributions **25_*Y***

(b) Total task failures for failure distributions **50_*Y***



(c) Makespan for failure distributions **25_*Y***



(d) Makespan for failure distributions **50_*Y***

Figure 11: Effect of failure distribution on the makespan of workflow and the total number of task failures in the system.

the GAS accordingly. On the contrary, in case of *Failure with Reputation*, the number of tasks scheduled by each GAS in the system is almost equal to that of *No Failure* as very few tasks are failed in this approach. For example, GAS14 schedules **100** tasks in *No Failure*, **102** in *Failure with Reputation*, whereas in case of *Failure without Self-adaptation*, it needs to schedule **216** tasks, which is **112%** greater than that in *Failure with Reputation* since **116** tasks, scheduled by GAS14 are failed by the Grid sites.

As the other GASs in the system also show the similar trend, the total number of tasks scheduled in the system, $\boldsymbol{SCH_{total}}$ for *Failure with Reputation* (**6569**) is much smaller than that for *Failure without Self-adaptation* (**8075**).

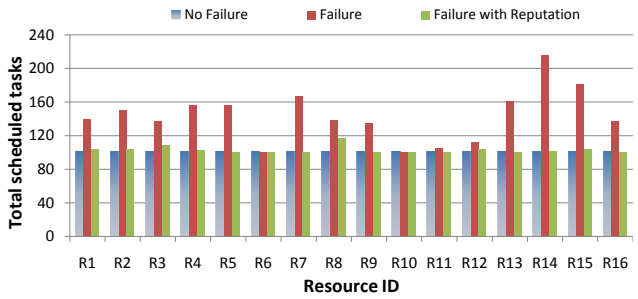


Figure 13: Total number of tasks scheduled by the GAS in the system (GAS1 - GAS16) for failure distribution **50_0.5**.

5.3.3. *Experiment 3: Measuring Pruning Efficiency*

Fig. 14 illustrates the pruning efficiency of the proposed scheduling technique. Fig. 14(a) and Fig. 14(b) show the total number of tasks successfully executed and failed by the resources in Grid site 1 to Grid site 16 respectively for **50_0.5**. From the figures, we can realize that in *Failure without Self-adaptation*, if a Grid site can execute task faster, it is assigned more tasks. Thus, the number of successful and failed tasks by that site is high if it's failure probability is low and high respectively.

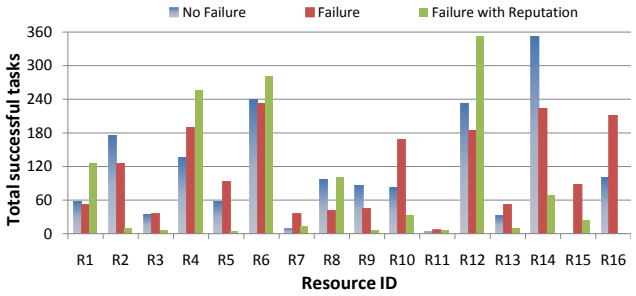
On the other hand, in case of *Failure with Reputation*, number of successful tasks by a Grid site is high if it is faster and does not fail any task. If it fails task, although it can execute task faster, it is not assigned any task further. Therefore, total failed tasks by that resource becomes very low. For instance, total failed tasks by resource R2 (with 0.59 failure probability and 3600 MIPS rating) is **152** in *Failure without Self-adaptation*, whereas it is only **11** in *Failure with Reputation*. Fig. 14(c) shows how failure-prone resource R2 is shunted out of the system over the period of time in our proposed reputation based scheduling approach.

5.3.4. *Experiment 4: Impact of Reputation Threshold*

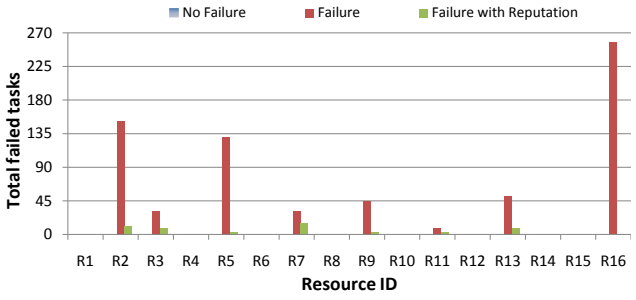
Fig. 15 shows the impact of reputation threshold ($\boldsymbol{R_{th}}$) on $\boldsymbol{F_{total}}$ and $\boldsymbol{M_{average}}$ in the system for *Failure with Reputation* when failure distribution is **50_0.5**. From the figure, it is evident that when $\boldsymbol{R_{th}}$ is slightly higher than **0**, $\boldsymbol{F_{total}}$ and $\boldsymbol{M_{average}}$ for *Failure with Reputation* are almost equal to that for *Failure without Self-adaptation*. This happens due to the reason that if $\boldsymbol{R_{th}}$ is very low then the reputation based scheduling scheme is

Table 8: Configuration for different experiments

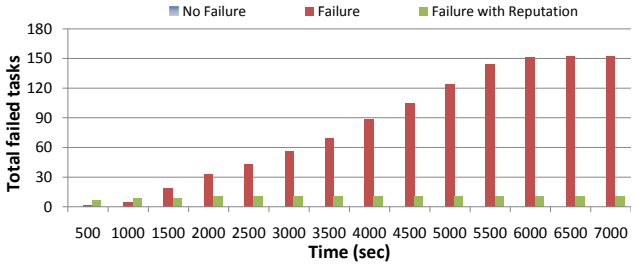
Parameter	Experiment 1.1	Experiment 1.2	Experiment 2	Experiment 3	Experiment 4	Experiment 5
n	64	64	64	64	64	64
no. of tasks	100	100 to 500	100	100	100	100
task size (MI)	50000 to 500000	50000 to 500000	50000 to 500000	50000 to 500000	50000 to 500000	50000 to 500000
failure distribution	25_0.1 to 25_0.9, 50_0.1 to 50_0.9	50_0.5	50_0.5	50_0.5	50_0.5	50_0.5
R_{th}	0.8	0.8	0.8	0.8	0.0 to 0.99	0.8
feedback function	exponential	exponential	exponential	exponential	exponential	exponential/simple



(a) Total number of tasks successfully executed by each resource (R1 - R16)



(b) Total number of tasks failed by each resource (R1 - R16)

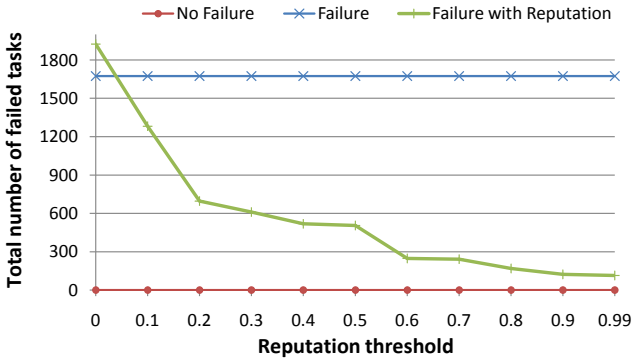


(c) Total number of tasks failed by Resource **2** over time

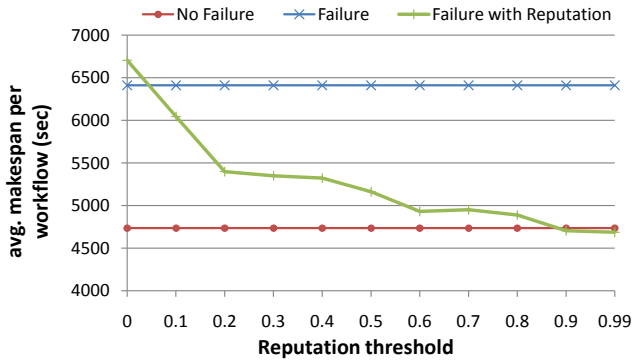
Figure 14: Effect of considering reputation on pruning failure-prone resources (failure distribution **50_0.5**).

not able to isolate the failure-prone resources with lower reputation score. Hence a considerable amount of tasks are assigned to those resources and F_{total} is increased eventually.

However, when the value of R_{th} is set a little bit higher than **0**, performance of the proposed reputation based scheduling approach in terms of scheduling efficiency is improved rapidly. Furthermore, with the increase of the value of R_{th}, average makespan and total task failures for *Failure with Reputation* gradually become almost equal to that for *No Failure*. For example, when R_{th} is set to **0.2**, **0.6** and **0.9**, F_{total} for *Failure with Reputation* is **58.4%**, **85.1%** and **92.6%** less than that for *Failure without Self-adaptation* respectively. Similarly, $M_{average}$ is also reduced by **15.8%**, **23.0%** and **26.6%** if R_{th} is set to **0.2**, **0.6** and **0.9** respectively.



(a) Total task failures vs. Reputation threshold

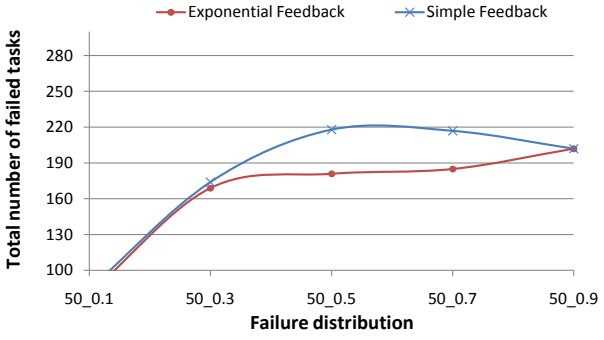


(b) Makespan vs. Reputation threshold

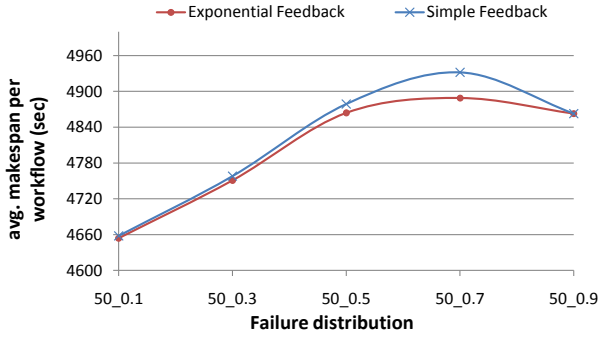
Figure 15: Effect of reputation threshold (R_{th}) on F_{total} and $M_{average}$ in the system for *Failure with Reputation* (failure distribution **50_0.5**).

5.3.5. Experiment 5: Performance of Exponential Feedback Function

Fig. 16 shows the significance of using exponential feedback functions on F_{total} and $M_{average}$ in the system when the proposed approach, *Failure with Reputation* is employed. In order to measure the performance, we compare our proposed feedback function against a simple linear feedback function available in the literature. From Fig. 16(a) and Fig. 16(b), it is evident that using an exponential function for calculating feedback results in reduced makespan and less number of total task failures in compare to using a simple linear feedback function. Although $M_{average}$ is not much varied, we can see a significant improvement in terms of F_{total}. For instance, in case of failure distribution **50_0.5**, when simple feedback function is used, **20%** more tasks are failed than using exponential feedback function.



(a) Total task failures



(b) Makespan

Figure 16: Significance of exponential feedback function on F_{total} and $M_{average}$ in the system for *Failure with Reputation* (failure distribution **50_*Y***).

5.4. Discussion and Summary

The results from the experiments show that there is a similarity of trend between the two performance metrics $M_{average}$ and F_{total}. Thus we have calculated the *Pearson's correlation coefficient* [3] and plotted the relationship between these metrics in Fig. 17.

Definition 5 (Pearson's correlation coefficient): Pearson's correlation coefficient $\rho_{P,Q}$ between two random variables P, Q with means μ_P, μ_Q and standard deviations σ_P, σ_Q is used to measure the linear relationship between them. It is defined as a quotient of the covariance of the two variables and the product of their standard deviations:

$$\rho_{P,Q} = \frac{cov(P,Q)}{\sigma_P \sigma_Q} = \frac{E((P-\mu_P)(Q-\mu_Q))}{\sigma_P \sigma_Q}$$

where, $\mu_P = E(P)$ and $\sigma_P{}^2 = E(P^2) - E^2(P)$.

The correlation is $\mathbf{1}$ when there is a positive linear dependence and $\mathbf{-1}$ in case of negative linear dependence. Zero indicates that there is absolutely no linear relationship between the variables.

The Pearson's correlation coefficient, $\rho_{M_{average},F_{total}}$ between $M_{average}$ and F_{total} in the system for different experiments conducted is listed in Table 9. From the table it can be observed that except for *Failure with Reputation* in *Experiment 1*, the values of $\rho_{M_{average},F_{total}}$ for both *Failure with Reputation* and *Failure without Self-adaptation* are greater than **0.9** in all other experiments. This indicates that there is a high degree of positive correlation or linear dependence between $M_{average}$ and F_{total} in the system. This happens due to the reason that when a task is failed, the task that depends on it's output needs to wait for longer period of time to be scheduled and executed. Therefore, the completion time of exit task is delayed and makespan of the workflow is increased, which indicates a linear relationship between $M_{average}$ and F_{total} (see Fig.17(b) - Fig.17(d)).

However, in case of *Failure with Reputation* in *Experiment 1*, workload is not heavy, F_{total} is small and failure-prone resources are isolated quickly. Thus makespan of the workflow is not increased over the resource failure probability although F_{total} is increase by a small margin. This means that there is no clear relationship or correlation between $M_{average}$ and F_{total} in such situation, which is reflected in Fig. 17(a).

Table 9: Pearson's correlation coefficient: $M_{average}$ vs. F_{total}

Approach	Exp 1.1 (25_Y)	Exp 1.1 (50_Y)	Exp 1.2	Exp 3
Failure with Reputation	0.2073	-0.4202	0.9904	0.9382
Failure	0.9673	0.9973	0.9971	-

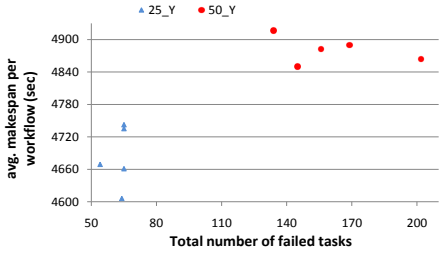
In summary, the above experimental and analytical studies indicate the following:

(i) considering reputation of Grid sites/resources for scheduling can increase the reliability of application scheduling in P2PG and improve the efficiency of distributed schedulers.

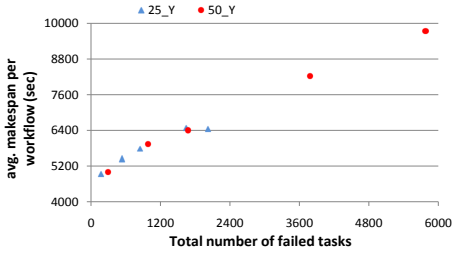
(ii) compared to *Failure without Self-adaptation*, *Failure with Reputation* can effectively reduce application completion time by avoiding potential task failures through intelligent scheduling irrespective of failure pattern of resources or workload on the system.

(iii) prunning efficiency of reputation based scheduling approach can be improved by increasing the reputation threshold in the system.

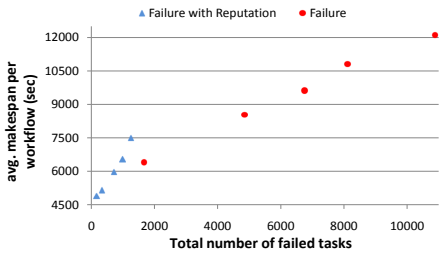
(iv) there is a high degree of positive correlation between



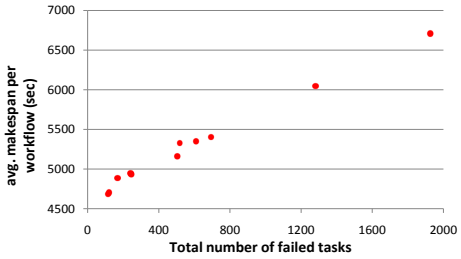
(a) Makespan vs. Total task failures for *Failure with Reputation*



(b) Makespan vs. Total task failures for *Failure without Self-adaptation*



(c) Makespan vs. Total task failures for varying workflow size (failure distribution **50._0.5**)



(d) Makespan vs. Total task failures for varying R_{th} (failure distribution **50._0.5**)

Figure 17: Correlation between F_{total} and $M_{average}$ in the system.

makespan of workflow and total task failures in the system.

6. Conclusion and Future Work

In this paper, we have presented a reputation based dependable scheduling technique for workflow applications in Peer-to-Peer Grids. Using simulation, we have measured the performance of the proposed scheduling technique against two cases: *Failure without Self-adaptation* and *No Failure*. The results show that our scheduling technique can reduce the makespan up to **50%** and successfully isolate the failure-prone resources from the system. Thus, by applying the proposed reputation based scheduling technique, not only context-aware and opportunistic placement of workflow tasks is possible but also significant performance gains are achievable (as analyzed in the previous section). Moreover, our results have practical importance since they highlight the fact that the schedulers, which do not have the ability to self-adapt in dynamic Grid conditions deliver degraded performance to application workflows.

Thus, it is reasonable to conclude that developing self-adapting Grid scheduling and application management techniques is important to exploiting the realm of Grids. Further, adapting to dynamic resource conditions aids in coping with the unpredictability and uncertainty of Internet-scale, multi-sites Peer-to-Peer Grids. In future, we intend to focus on implementing this reputation based dependable scheduling technique in real world P2PG system such as Aneka Federation [25]. As this paper shows that the variation in R_{th} has an impact on the system performance, in our future work, we also endeavour to devise an approach considering dynamic R_{th}, adjusted by the scheduler.

7. Acknowledgements

This work is partially supported by Australian Research Council (ARC) Discovery Project grant. We gratefully thank Xiaofeng Wang for his assistance in formulating the distributed reputation model.

[1] M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen, M. Parashar, B. Khargharia, and S. Hariri. AutoMate: enabling autonomic applications on the grid. In *Proceedings of Autonomic Computing Workshop*, USA, June 2003.

[2] F. Azzedin and M. Maheswaran. Integrating Trust into Grid Resource Management Systems. In *Proceedings of 31st International Conference on Parallel Processing, Canada*, 2002.

[3] N. Balakrishnan and C. R. Rao. Order statistics: Applications. *Handbook of Statistics, vol. 17*, 1998.

[4] P. Blaha, K. Schwarz, G.K.H. Madsen, D. Kvasnicka, and J. Luitz. Wien2k - an augmented plane wave plus local orbitals program for calculating crystal properties. Technical report, Vienna University of Technology, Austria, 2001.

[5] B. Bode, D. Halstead, R. Kendall, and D. Jackson. PBS: The portable Batch Scheduler and the Maui scheduler on Linux clusters. In *Proceedings of 4th Linux Showcase and Conference, Atlanta, USA*, October, 2000.

[6] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience, Volume 14, Issue 13-15, pages 1175-1220, Wiley Press*, 2002.

[7] E. Byun, Y. Kee, E. Deelman, K. Vahi, G. Mehta, and J. Kim. Estimating resource needs for time-constrained workflows. In *Proceedings of 4th IEEE International Conference on eScience*, USA, December, 2008.

[8] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. Introduction to algorithms. *MIT Press*, 1990.

[9] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, USA, June, 2001.

[10] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. H.

Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflow onto the grid. In *Proceedings of Across Grids Conference, Cyprus*, 2004.

[11] L. Clementi et al. Services oriented architecture for managing workflows of avian flu grid. In *Proceedings of 4th IEEE International Conference on eScience*, USA, December, 2008.

[12] T. Fahringer et al. Askalon: A tool set for cluster and grid computing. *Concurrency and Computation: Practice and Experience, 17:143-169*, 2005.

[13] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing*, USA, June, 2001.

[14] P. Garca, C. Pairot, R. Mondjar, J. Pujol, H. Tejedor, and R. Rallo. Planetsim: A new overlay network simulation framework. In *Proceedings of Software Engineering and Middleware, Linz, Austria*, 2004.

[15] W. Gentzsch. Sun Grid Engine: Towards Creating a Compute Power Grid. In *Proceedings of 1st IEEE International Symposium on Cluster Computing and the Grid, Brisbane, Australia*, May, 2001.

[16] A. Iosup, Hui Li, Mathieu Jan, Shanny Anoep, C. Dumitrescu, Lex Wolters, and Dick Epema. The grid workloads archive. *Future Generation Computing Systems, Elsevier Press, Amsterdam, The Netherlands*, 2009.

[17] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proceedings of 12th international conference on World Wide Web*, Hungary, 2003.

[18] D. Kececioglu. *Reliability Engineering Handbook*. Prentice Hall, Inc., New Jersey, Vol. 1, 1991.

[19] J. Kim, P. Keleher, M. Marsh, B. Bhattacharjee, and A. Sussman. Using content-addressable networks for load balancing in desktop grids. In *Proceedings of 16th international symposium on High performance distributed computing*, USA, June, 2007.

[20] M. Litzkow, M. Livny, and M. Mutka. Condor-a hunter of idle workstations. In *Proceedings of 8th International Conference of Distributed Computing Systems, IEEE CS Press, USA*, June 1988.

[21] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience, Special Issue on Scientific Workflows*, 2005.

[22] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics, 20(17):3045-3054*, 2004.

[23] M. Rahman, R. Ranjan, and R. Buyya. Cooperative and decentralized workflow scheduling in global grids. *Future Generation Computing Systems (in press), Elsevier Press, Amsterdam, The Netherlands*, 2010.

[24] L. Ramakrishnan, M. Reed, J. Tilson, and D. Reed. Grid portals for bioinformatics. *Renaissance Computing Institute, University of North Carolina, USA*.

[25] R. Ranjan and R. Buyya. *Decentralized Overlay for Federation of Enterprise Clouds*. Handbook of Research on Scalable Computing Technologies. K. Li et al. (eds), IGI Global, USA, 2009.

[26] R. Ranjan, L. Chan, A. Harwood, S. Karunasekera, and R. Buyya. Decentralized resource discovery service for large scale federated grids. In *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing, India*, December, 2007.

[27] R. Ranjan, A. Harwood, and R. Buyya. A case for cooperative and incentive based coupling of distributed clusters. *Future Generation Computer Systems, Volume 24, No. 4, Pages: 280-295, Elsevier Press, Amsterdam, The Netherlands*, 2008.

[28] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, USA, 2001.

[29] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[30] F. Schuller and J. Qin. Towards a workflow model for meteorological simulations on the austriangrid. In *Proceedings of 1st Austrian Grid Symposium, Schloss Hagenberg*, Austria, December, 2005.

[31] J. Sonnek, A. Chandra, and J. Weissman. Adaptive Reputation-Based Scheduling on Unreliable Distributed Infrastructures. *IEEE Transactions on Parallel and Distributed Systems, volume 18, issue 11, pages 1551-1564*, 2007.

[32] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM Conference on Applications, technologies, architectures, and protocols for computer communications*, USA, 2001.

[33] E. Tanin, A. Harwood, and H. Samet. A distributed quad-tree index for peer-to-peer settings,. In *Proceedings of 21st IEEE International Conference on Data Engineering, Tokyo, Japan*, 2005.

[34] I. Taylor, M. Shields, and I. Wang. Resource management of triana p2p services. *Grid Resource Management, Netherlands*, June 2003.

[35] D. Theiner and P. Rutschmann. An inverse modelling approach for the estimation of hydrological model parameters. *Journal of Hydroinformatics*, 2005.

[36] H. Topcuouglu, S. Hariri, and M.Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems, volume 13, issue 3, pages 260-274*, 2002.

[37] Y. Yang, J. Chen, J. Lignier, and H. Jin. Peer-to-peer based grid workflow runtime environment of swindew-g. In *Proceedings of 3rd IEEE International Conference on e-Science and Grid Computing, India*, December 2007.

[38] J. Yu and R. Buyya. A novel architecture for realizing grid workflow using tuple spaces. In *Proceedings of 5th IEEE/ACM Workshop on Grid Computing, IEEE CS Press, USA*, 2004.

[39] R. Zhou and K. Hwang. PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing. *IEEE Transactions on Parallel and Distributed Systems, volume 18, issue 4, pages 460-473*, 2007.