I.J. Information Engineering and Electronic Business, 2015, 6, 53-62
Published Online November 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2015.06.08

Improving Fault-Tolerant Load Balancing Algorithms in Computational Grids

Jasma Balasangameshwara
Atria Institute of Technology, Karnataka, India
Email: jasma2002@gmail.com

Abstract—Fault tolerant scheduling of many jobs in an environment with millions of unpredictable nodes is not an easy issue. To the best of our knowledge, no work in the literature has proposed a solution that combines the merits of active and passive replication schemes of fault tolerance with the advantages of performance-driven load balancing so as to make the most of the strong points of each. While extensive fault tolerant scheduling and load balancing methods have been presented for the sequential jobs, none have taken into account fault-tolerant load balancing that minimizes the job make-span, provides efficient network and node utilization, achieves a Ill-balanced load and high system flexibility even during the resource failures. Hence, in this article, I present an *Adaptive Scheduling Algorithm* namely ASA that overcomes these problems. With thorough simulations, I conclude that ASA allocates any number of jobs to a million nodes with relatively low overhead and high flexibility. Experimental results show that the performance of ASA is better than those of its counterparts.

Index Terms—Computing Grid, Fault-Tolerant Scheduling, Load Balancing.

I. Introduction

Due to the growth of science and engineering, problems in these fields have become complicated. To solve such complex problems, a dominant computing facility is required. A computing grid is a federation of hardware and software infrastructures from various locations that offer reliable, consistent, persistent and economical access to high-end computational capabilities [1]. They enable active sharing, aggregation and selection of geographically dispersed, autonomous and diverse nodes at run-time based on their availability, performance and capability.

In a real world scenario, the job arrival patterns are volatile and the computing capabilities erratic and asymmetrical. The nodes in a particular grid site may become overloaded while other grid sites may be under-loaded [2]. Therefore, the heterogeneous and the dynamic environment of grids require load balancing and fault-tolerance in order to make the best usage of the performance of the grid nodes.

1.2 Background

Job scheduling is an efficient approach to realize high application performance in grid environments. Job scheduling algorithms are divided into two types: *batch mode* and *on-line mode*. In *batch mode*, jobs are queued and collected into a set. The scheduling algorithm initiates after a fixed time period. Batch mode algorithms are more suitable for environments utilizing the same resource. In *on-line mode*, jobs are scheduled when they arrive. Since the grid environment is heterogeneous and processor speeds vary among nodes, on-line mode scheduling is more appropriate for the grid environment [3]. Our work is focused on non-pre-emptive online scheduling of a-periodic and sequential jobs.

Load balancing plays an imperative role in improving grid node utilization and reducing the usage of time. Also, the method used for load balancing affects the performance of the grid system. Hence, load balancing techniques should be "fair" in distributing the load across the grid nodes. The meaning of "fair" is that the difference between the "heaviest loaded" node and "lightest loaded" node should be minimized [4].

In a grid system, resource failures come about frequently leading to damaging consequences. Hence, I need to develop efficient methods to tolerate such failures. Fault tolerance can be classified as [5]

(1) *Active replication mode*: This method is based on space redundancy and does not require fault detection.
(2) *Passive replication mode*: In this method, a backup copy of a job is activated only if a fault occurs while executing its primary copy. The techniques applied while scheduling primary and backup copies of a job are: (1) *Backup Overloading:* This consists of scheduling backups for multiple primary jobs during the same time slot. (2) *De-allocation* of resources reserved for backup jobs when the corresponding primaries complete successfully.

1.2 Motivation

There have been huge efforts in the recent years for developing fault-tolerant scheduling and load balancing algorithms to realize high job performance in a grid environment. Unfortunately, many practical instances of such problems are found to be NP-complete [2]. Hence, our work is motivated by the need for the competent techniques that takes into consideration grid architecture, resource and job heterogeneity, communication delay and

resource volatility. Our foremost intention is to arrive at job assignments that realize minimum job make-span, efficient network and node utilization, an Ill-balanced load, high system reliability and flexibility even during the resource failures. A resource failure can be a link and/or processor failure.

Many fault-tolerant scheduling techniques and load balancing methods have been designed for computing grids supporting sequential jobs. However, to the best of our literature understanding, this study is the first of its kinds to jointly consider hybrid fault-tolerant scheduling and load balancing for sequential jobs. It is a challenge to propose a new hybrid fault-tolerant scheduling and load balancing strategy for sequential jobs running on a computing grid. This challenge is the motivation to extend the scheduling method proposed in [6] and integrate it with fault-tolerant load balancing approach [7] for sequential jobs.

1.3 Contribution

My contributions in this article are multifold. I have proposed a hybrid fault-tolerant load balancing technique for a computing grid. Given that, grid infrastructures are dynamic in nature with a varying network topology; I generate a random topology with nodes of varying capacities and varying bandwidth between the links connecting them.

To the best of our knowledge, I are the first to cumulatively tackle the following:

1. Juxtaposing the merits of active replication and passive replication schemes of fault-tolerance, the advantages of performance-driven load balancing to form a hybrid one so as to make the most of the strong points of each.
2. ASA takes into account grid architecture, resource and job heterogeneity, communication delay and resource unpredictability for sequential jobs.
3. ASA minimizes the communication cost and replication cost of sequential jobs.
4. ASA aims to deal with millions of nodes with varying frequency of failures.
5. I have conducted trace-driven simulation tests using a million nodes.

The key contributions of this study to the related existing work are summarized as follows:

1. Mutual information feedback scheme and neighbor selection strategy proposed in [7] are further enhanced.
2. Estimating the availability and efficiency of nodes by simplifying the modeling of decision making in fault-tolerant scheduling.

Simulation results show that high job performance gains are achieved in terms of jobs make-span.

Remainder of this work is organized as follows: Section 2 reviews related work in literature. The overview of the system model is presented in section 3. Section 4 presents our proposed algorithm in detail. Section 5 focuses on the setup of the simulation and the experimental results. Finally, section 6 is dedicated to conclusion and future work.

II. Related Work

Despite the facts that load balancing, job scheduling and fault-tolerance are active exploration areas in grid environments, these areas have more often than not been and continue to be developed independently of one another, each focusing on diverse aspects of computing.

It has been shown that as more information is collected by an algorithm, the algorithm can make better load balancing decisions however at an increased overhead [16]. Dynamic load balancing technique is opted for heterogeneous systems in computational grids because the technique considers nodes with different processing speeds and memory sizes, bandwidth and load [16].

With respect to load balancing and job scheduling, Lu et al. [10] proposed a distributed load balancing technique for a computing grid which took into account the issues of scalability, resource heterogeneity and significant communication overhead. However, Lu did not consider the execution scheme for data distribution. Braun et al. [11] proposed Minimum Execution Time (MET) and Minimum Completion Time (MCT) scheduling methods. But both MET and MCT leads to poor make-span. The Most Fit Task First (MFTF) method proposed in [12] discovers the fitness between jobs and nodes. However MFTF does not consider node utilization and the estimation function is an ideal function leading to incorrect scheduling in real environment. Lee et al. [13] proposed hierarchical job scheduling framework for load balancing and minimizing the job make-spans. However, Lee neglected job length and assumed the system to be static.

Rafiqul and Ali proposed a diffusion load balancing algorithm for distributed computing systems [16]. They have shown that the communication overhead depends on the load of the nodes. However, they have not considered the dynamics of communication delay incurred due to distance among the nodes while making a load balancing decision.

Faulty nodes in a distributed environment can lead to prolonged job waiting times and have a major impact on the load balancing performance. Hence, for improving the throughput of a system it is important to make the system fault-tolerant.

Many fault-tolerant methods have been proposed for grid systems [6], [9] and [14]. Luo et al. [14] proposed DYFARS for heterogeneous systems. DYFARS considers both active and passive replication methods thereby proving high system flexibility. However, it did not substantiate its work with realistic workload and large number of nodes. Zhu et al. [6] proposed fault-tolerant scheduling algorithm for heterogeneous clusters. However, they focused on a centralized scheduling model and did not take into account network bandwidth while scheduling real-time tasks.

Danial et al. proposed a distributed fault tolerant algorithm for distributed computing systems for eliminating single point of failure [17]. A major advantage of the algorithm is that it exchanges fever messages as compared to ring algorithm, modified Bully algorithm. However, the work does not consider optimizing nodes for message transfer.

In this article, I pay interest on the issues of hybrid load balancing and non-pre-emptive fault-tolerant scheduling of sequential jobs for a computing grid. Given a dynamically changing load, *ASA* adaptively switches between passive replication scheme and active replication scheme. To achieve high scheduling ability and an Ill-balanced load, ASA optimizes the resource and network utilization for sequential jobs.

III. System Model

In this section, I bring in the models, concepts and terminology used in this article. The model presented in this article provides a scalable fault-tolerant load balancing service to nearly any type of a computing grid platform.

3.1 Scheduling Model

The scheduling model and node model addressed in this article are based on the scheduling model and node model outlined in [8]; with similar ideas and terminologies as in [8].

The distributed scheduling model that has been proposed in this article is divided into two models namely a local model and a global model. The local model describes the working of the local scheduler. The local scheduler runs on each processing node, to control the processing of jobs in its job queue. The global model describes the availability status, the efficiency information and the forwarding algorithm. The global scheduler determines the implementation of these three elements.

Each local scheduler computes the availability of its node to execute the jobs using an event-driven policy and exports this information. The availability status is then used by the forwarding algorithm to route the jobs to the most appropriate processing node.

3.2 Node Model

Physical nodes are structured in a logical overlay manner. The nodes are connected via different communication links with different speeds. In this overlay structure; a node c_i may take part in any of the following roles:

1. The processing node P_i provides a local scheduler and a scheduling environment for the processing of the jobs.
2. The submission node S_i is in charge of the assignment and monitoring of the jobs.
3. The routing node R_i is a part of the global model. It distributes the availability status, efficiency information and forwards the jobs.

Nodes can play all the three roles for load balancing purpose. The roles played by the same physical nodes are placed independently within the overlay. The processing environment at each processing node P_i allows the node owner to arbitrarily limit the type and amount of resources that a job may use.

1. The processing power PP_i is measured in million instructions per second (MIPS).
2. The available memory reserved for processing a job AM_i is measured in megabytes.
3. The available disk space AD_i is also measured in megabytes.

To account for different network topologies, a random connection graph is generated with a definite set of nodes. Initially, a minimum spanning tree with all the nodes is generated and then arbitrary links are added to the tree so as to reproduce the final grid topology. This approach not only gives flexibility on the number of links and the grid topology but also allows the rearrangement of any of the roles played by a node without affecting the other ones.

3.3 Fault Model

In our model, the node availability information is distributed in a reactive manner. Hence, any omitted information is rapidly recovered by the neighbor nodes in case of failures [8]. The nodes deal with the failures by setting timeouts so as to discover lost and aborted jobs [8]. Nodes also deal with their own failures by saving the state of their submitted jobs into a database.

The global scheduler GS_i together with the set of nodes that GS_i interacts with is called a group. A node may belong to more than one group. The fault model outlined below is employed in each group [6], [7] and [8].

- Faults can be transient or permanent and are independent.
- The failure may occur in the hardware, operating system, grid middleware module or in the network connection.

When the primary job finishes its execution, its backup is deleted so that the backup slot can be freed timely for new jobs. This process is called resource reclaiming. In passive backup-copy mode, the backup jobs can overload as long as their primaries are scheduled on different processors.

3.4 Job Model

I consider a set of sequential jobs $J= \{j_1,j_2,\ldots,j_n\}$ of computationally intensive, independent, non-pre-emptive, real-time and a-periodic jobs with no required order of execution. The jobs are of different computational sizes and have different input and output size requirements. Each node is characterized by its processing capacity, which is defined as the product of CPU speed and the idle CPU percentage. Furthermore, all the jobs in the job queue JQ (P_i) are prioritized by their arrival time, hence there is only a single job being executed by a node P_i at a given

time while other jobs are waiting in the job queue. The global scheduler is responsible for inquiring the present states of all nodes in its group and then the execution time of the jobs is estimated accordingly.

The job model addressed in this article is based on the task model outlined in [6]; with analogous notions and terminologies as in [6]. Since I are using primary-backup approach, each job j_i has two copies j_i^P and j_i^B executed on primary and backup nodes respectively.

Let $PR_i=(a_i,d_i,e_{ik},am_i,ad_i,\ldots)$ be the tuple that describes the job j_i. Given a job j_i belonging to J, I denote the arrival time as a_i, deadline as d_i, execution time of job j_i on processing node P_k as e_{ik}, required memory as rm_i, and required disk space as rd_i. This configuration is very common in a grid scheduling platform due to its high degree of parallelism [8].

Let f_i^P and f_i^B be the finish times of the primary copy j_i^P and the backup copy j_i^B respectively. Let l_i^P denote the laxity of j_i^P i.e. $l_i^P=d_i-f_i^P$.

X_i^P and X_i^B represent all possible schedules for primary copy j_i^P and backup copy j_i^B respectively. $x_i^P \in X_i^P$ is a scheduling decision of j_i^P. Similarly $x_i^B \in X_i^B$ is the scheduling decision of j_i^B. $x_i^P$and x_i^B are feasible scheduling decisions if $f_i^P \leq d_i$ and $f_i^B \leq d_i$.

Let $e_k(x_i^P)$ and $e_k(x_i^B)$ denote the execution time of job j_i^P and j_i^B on node P_k using scheduling decision x_i^P and x_i^B respectively. Recollect that backup copies may be active or passive. ASA adaptively decides the backup-copy mode based on the laxities of the primary copies.

Let m_i^B denotes the backup-copy mode of the job j_i^B. The back-up copy mode can be expressed as in equation 1.

$$m_i^B = \text{passive If } l_i^P > e_k(x_i^B) \text{ else active} \quad (1)$$

3.5 Hybrid Load Balancing Model and Scheduling Principles

The load $LD_{i,t}$ of a node P_i at a particular instant of time t is estimated by the weighted sum of squares method. I take four parameters into account; network utilization, memory utilization, idle CPU percentage and length of jobs in $JQ(P_i)$ divided by the processing capacity of the node.

The instantaneous policy uses First-Come-First-Served (FCFS) to find the earliest completion time of each job independently based on the information provided by the global scheduler. The instantaneous policy decides whether the job has to be executed locally or sent to its neighbors. This choice depends on whether the job gets a performance gain if it is routed to a neighboring node (neighbor selection and performance gain function are discussed in section 4).

The following principles are adopted by ASA:

- ASA realizes job assignments that accomplish minimum response time and optimal node utilization.
- Given the same execution time, ASA schedules jobs on the nodes with low failure rates.
- For a group of nodes with the same failure rate, ASA assigns jobs in order to the nodes that offer least load.

IV. Adaptive Scheduling Algorithm

4.1 Group Creation Method

The group creation method considered in this article is based on the neighbor selection policy outlined in [7]; with similar notions and terminologies as in [7]. The neighbors for a node are formed in terms of communication cost. Communication cost is determined using weighted sum of squares method. It has two parameters; data transmission rate and bandwidth [7]. They are computed as given in [7]. For a global scheduler GS_i, a global scheduler GS_k is considered as its neighbor global scheduler as long as the communication cost between GS_i and GS_k is within β times the communication cost between GS_i and the nearest global scheduler and the load of GS_k is less than GS_i. The distance coefficient β=1.375 yields superior results (the variations of this random value for experiments are provided later).

A group is a subset of G. The group $group_i$ of a global scheduler GS_i includes the global scheduler GS_i and its neighbors. Global scheduler updates load based on the information collected in the last exchange interval.

4.2 Node Efficiency Estimation Technique

The efficiency estimation technique addressed in this article is based on the resource efficiency estimation policy outlined in [7].

Each global scheduler maintains a Schedule List (SL) to record a schedule. It deletes the recorded schedule from the SL if the job is completed. The global scheduler records the schedule information of a job j_i as a tuple PR_i with additional parameters as node P_q, estimated completion time ect_i and job size js_i in million instructions.

If a global scheduler has not at all scheduled jobs to a node, it knows nothing about it. Each global scheduler creates a count based on the jobs states in the SL. If a schedule is successfully finished then the count is set to -1 else the count is +1. If numerous jobs are scheduled to the same processing node, then an independent count is assigned to each schedule. Finally, the fitness of a processing node is computed by summing up all the counts. The efficiency of a processing node is as shown in equation 2.

$$(k)^i.effi += h(P_q) \quad (2)$$

Where $(k)^i.effi$ is the efficiency of k number of schedules on the processing node P_q maintained by global scheduler GS_i and $h(P_q)$ is the fitness of processing node P_q. A processing node is said to be the most efficient if it has the smallest efficiency value.

4.3 Availability Status Estimation Technique

The availability status is the join between the local scheduler and the forwarding algorithm [8]. The

availability function describes the availability of a processing node to accept new jobs. It is defined as follows:

Definition 1: The availability function $AF_i(PR_u)$ describes the availability of the processing node P_i at the current time with the job parameters PR_u that P_i is able to process [8].

The availability function for P_i would be $AF_i(rm_u,rd_u)$. For any pair of job parameters (rm_u,rd_u) such that $rm_u \leq AM_i$ and $rd_u \leq AD_i$, the function returns a value of 1 else 0. Processing nodes send their availability to the routing nodes, thereby providing their state to the forwarding algorithm. The forwarding algorithm decides as to how to distribute the jobs among the routing nodes.

Property 1. A group can tolerate one computing node's failure if and only if the job's primary copy and the backup copy are scheduled on two different computing nodes [7].

Property 2. The earliest start time est_{ij}^P of a primary copy j_i^P on the processing node P_j must satisfy the following constraints [6]:

- Processing node P_j has an idle time slot adequate enough to hold j_i^P
- The finish time of j_i^P must be less than or equal to the deadline of j_i.

Before est_{ij}^P can be computed, it is assumed that a set of jobs $j_1,j_2,\ldots,j_q$ have been allocated to node P_j. These jobs can be either primary copies or backup copies. Hence, the idle time slots on P_j can be represented as $[0,s_1],[f_1,s_2],\ldots,[f_{(q-1)},s_q],[f_q,\infty]$. To get est_{ij}^P, all idle time slots are scanned from left to right. The first idle time slot $[f_k,s_{(k+1)}]$ that satisfies the following is chosen as shown in equation 3:

$$s_{(k+1)}-\max(a_k,f_k) \geq e_{ij}(x_i^P) \quad (3)$$

Thus, the earliest start time est_{ij}^P of j_i on p_j is determined as shown in equation 4.

$$est_{ij}^P = \max(a_k,f_k) \quad (4)$$

To efficiently utilize the grid system resources, the backup copy of a job needs to employ passive replication mode. If a backup copy adopts active replication mode, then its processing time should be as less as possible. Hence, in our model, the primary copy of a job is scheduled as early as possible and the backup copy of the corresponding job is scheduled as late as possible. A similar model can be found in [6].

4.4 Performance Gain Function

The performance gain function (PBF) considered in this article is analogous to that outlined in [7]. Jobs are allocated to processing nodes with enough memory and disk space.

$\forall GS_i \in G$, the transfer cost of sending a job $j_x \in J$ from GS_i to GS_q at time instant t is estimated by equation 5.

$$tc(j_x,GS_i,GS_q,t) = \quad (5)$$

$$TransIn(j_x,GS_i,GS_q,t)+LD_q+(k)^q.effi+TransOut(j_x,GS_q,GS_i,t)$$

$TransIn(j_x,GS_i,GS_q,t)$ and $TransOut(j_x,GS_q,GS_i,t)$ measure the time taken to transfer j_x from GS_i to GS_q and GS_q to GS_i respectively. The performance gain function PBF_x for a job j_x is calculated as shown in equation 6.

$$\forall GS_i \in GGS_i \neq GS_q$$
$$\text{If } AF_q \text{ returns 1 and Property 2 is satisfied then} \quad (6)$$
$$PBF_x = tc(j_x,GS_i,GS_i,t)-tc(j_x,GS_i,GS_q,t)$$

The performance gain function for a job j_x is based on the thought that j_x can be allocated to a processing node that would gain most in terms of the expected response time [7].

4.5 Distribution of Availability Status and Efficiency Information

Processing node availability and efficiency changes frequently, at least every time a job starts or finishes. Hence, the availability status and efficiency information must be kept up to date at the routing nodes. This involves deciding how and when to propagate the updates information. Hence, in our work, I have used the mutual information feedback policy given in [7] to propagate the updated information.

4.6 Efficient Forwarding Algorithm

I have further improved the mutual information feedback policy (MIF) presented in [7] by taking into account network performance, making the job transfer request (JTR) and job completion reply (JCR) small sized and reducing the overhead induced due to timestamp comparisons during the job transfers. Each global scheduler GS_i maintains the state information of its neighbors in a state object Y^i. Each item in $Y^i[k]$ has a property list (load, efficiency, idle time slots, time). $Y^i[k].load$, $Y^i[k].efficiency$, $Y^i[k].idleTimeSlots$, $Y^i[k].time$ denotes the load, efficiency, list of idle time slots and the local time when the load, efficiency and idle time slots information respectively of the global scheduler GS_k is reported. The MIF policy employs piggy-backing strategy. The global scheduler GS_i will send JTR to GS_k only if GS_k is available and property 1 and 2 are satisfied. When GS_i sends a JTR to GS_k for processing for the first time, GS_i appends to the JTR the state information of itself and its neighbors. If GS_k is updating its state object for the first time, then it compares the timestamps to check if the global schedulers contained in the JTR received from GS_i belong to its neighbors.

If the JTR from GS_i to GS_k is not for the first time, then GS_i appends to the JTR the state information of itself, its and GS_k's common neighbors so that GS_k can update its state object and vice versa. In order to surmount the overhead induced on the network bandwidth due to frequent job migration, the JTR carries the source grid scheduler's information to the destination. Thus whenever

the JTR reaches its target, the job can be downloaded from the source as required [7].

4.7 Load Balancing Policy

The load balancing policy is triggered whenever the global scheduler GS_i receives state information from its neighbors. The policy will use the most recent state information to route the job. Each global scheduler without violating property 1 and property 2 submits some jobs to one of its neighbor which has minimum load among all. If all its neighbors are busier than itself, no job is submitted from that node. In exchanging load from a heavier loaded node to lighter loaded node, attention is given not to violate property 1 and property 2 and not to burden the lighter node such that it exceeds the load of the second lighter node among its neighbors. Neighbors of c_i are sort in ascending order based on the load.

4.8 Primary Copy Allocation

Consider a job $j_i \in J$ with j_i^P and j_i^B as its primary copy and backup copy respectively. To make the backup copy j_i^B employ passive replication mode, its corresponding primary copy j_i^P should be scheduled as early as possible on a processing node P_u. If j_i^P is allocated to P_u, then it must finish within its deadline. If j_i^P can be scheduled on numerous nodes, then a node with maximum performance gain is selected. If some nodes make the system have identical performance gain, then the node on which the start time of j_i^P is earliest is selected. If j_i^P cannot be finished within its deadline on any nodes, then j_i^P it is rejected.

4.9 Distributed Fault Tolerant Model

The distributed fault tolerant model employed by ASA is as follows: If a node shuts down manually, it sends a notice message to its backup and its other neighbors before shutting down. ASA uses a peer-to-peer fail-over strategy such that each node is a backup of another neighbor node in a grid system. In the present implementation, for a primary node, I have chosen those backup nodes that completes the jobs quickly and have nominal replication cost among other neighbors [7].

Replication cost is defined as the actual percentage of time needed for scheduling the backup copy in addition to all the overloaded periods with the existing backup copies. Scheduling the backup of a job where its start time and/or finish time collides with the boundaries of interval or boundaries of over-loadable backup schedules is referred to as a boundary schedule. All the neighbor nodes in addition to the one where the primary copy is scheduled on the boundary schedules within the time window are considered and their replication cost is compared. The boundary schedule which has the least replication cost is chosen. A schedule is said to be qualified if it is contained in the time window and does not overlie on any primary schedule or non-over-loadable backup schedule. In particular, there must not exist any primary schedule or non-over-loadable backup schedule that starts and/or ends in this schedule or contains this schedule [9].

4.10 Backup Copy Allocation

To make j_i^B employ passive replication mode, j_i^B's execution begins as late as possible. The latest start time lst_{ij}^B of job j_i^B on the processing node P_j is calculated as shown in equation 7 [6]

$$lst_{ij}^B = \begin{cases} st_x - e_{ij}(x_i^B) & \text{iff } d_i \geq st_x \\ d_i - e_{ij}(x_i^B) & \text{else} \end{cases} \quad (7)$$

where st_x denotes the start time of job j_x that is scheduled following j_i^B and it cannot overlap with j_i^B. If j_i^B can be scheduled on P_j, then j_i^B should satisfy property 2 and finish within its deadline. If the latest start time of j_i^B is later than or equal to the finish timeof j_i^P, then j_i^B is capable of employing passive replication mode. Hence, j_i^B need not use active replication mode on other nodes any more. If j_i^B can execute using active replication mode on some nodes, then the node which gives maximum PBF is selected. If the lastest start time of j_i^B is earlier than the finish time of j_i^P, then j_i^B executes using active replication mode. In this circumstance, the concurrent processing time of j_i^P and j_i^B should be as little as possible. Hence, the node on which j_i^B has latest start time is selected. If j_i^B cannot satisfy the timing constriant or property 2 on any nodes, both j_i^P and j_i^B are rejected.

Theorem 1. The time complexity of ASA is O(kq), where k is the number of processing nodes in the grid system and q is the number of waiting jobs on the processing node.

Proof: To obtain est_{ij}^P and lst_{ij}^B of j_i^P and j_i^B respectively on some node P_j, the time complexity is O(q). The time complexity to calculate the system reliability after the job is allocated to P_j is O(1). Hence the time complexity of ASA is calculated as O(k)(O(q))=O(kq).

The theorem shows that the operation time of the algorithm is proportional to the number of processing nodes and the number of waiting jobs on a processing node.

V. Brief Description of PD_MinRC

To demonstrate the performance improvements gained by ASA, I contrast it with PD_MinRC [7].

PD_MinRC juxtaposes the strong points of neighbor-based and cluster-based load balancing methods. Neighbors for a resource are formed in terms of transfer delay. The distribution and location policies are performance-driven to minimize replication cost and communication cost. The communication overhead involved in information collection is reduced by using enhanced MIF policy where the job transfer request is made simple and small-sized. PD_MinRC also takes into account the resource efficiency and dynamic resource failure in a grid.

However, PD_MinRC ignored data transmission rate during neighbor selection. In MIF policy of PD_MinRC, each time a resource updates its state object, it comparesits timestamps. ASA eliminates the overhead induced due to frequent timestamp comparisons thus making the JTR

even more simple and small-sized. PD_MinRC does not consider the memory and disk space requirements of a job which is taken care by ASA. PD_MinRC employs passive replication scheme while ASA adaptively switches between passive and active schemes. Load and communication cost in ASA are computed using the weighted sum of squares method. PD_MinRC is tested using just 500 sites with one node at each site. ASA deals with millions of nodes with varying frequency of failures. PD_MinRC assumes a definite distribution but ASA uses real-time trace-driven workloads of many years.

VI. Experimental Results

The scalability, performance and fault-tolerance of ASA is measured through a set of simulations and tests. Detailed tests have been considered to assess the accurateness of the ad-hoc simulator written in java, so as to monitor ASA under practical conditions. The ad-hoc simulator minimizes the memory constraints so as to create as many nodes as possible.

6.1 Simulation Setup

The simulation parameters used in our work are analogous to those used in [8].

The ad-hoc simulator models the behavior of a network of nodes with direct communication channels among each pair of nodes. The network is of configurable size. To check the scalability of ASA, networks from five thousand up to a million nodes have been simulated. Since, both ASA and PD_MinRC have low complexity, I Ire able to simulate up to a million nodes. For each generated job, the simulator considers the transmission delay and the computation times. The transmission delay of a job is computed by modeling the end-to-end link with 50Mbps bandwidth and a delay ranging between 50ms to 500ms, and of a fast cluster interconnection network of 2Gbps bandwidth with a delay of 1ms to 2ms.

The job processing time is calculated using the computing power of the processing nodes. There is a user at each node endlessly submitting new jobs. The traces of the workload contain activity of thousands of users with information of millions of jobs taken from many years [8]. This information is also used for extracting the distribution of computing power, job's computational requirements, memory, disk space, mean time between failure and mean time to recover. The deadline distribution is computed as $D=\omega*(0.8/U-1)/0.05$ where U is the standard uniform distribution, where ω is a parameter less than 1.

In ASA, while calculating the load of each node, the weighted values of the corresponding load attributes are considered. The load of each node is computed by considering memory utilization, network utilization, idle CPU percentage and length of jobs in the job queue by the processing capacity of the node. The weighted value corresponding to memory utilization is w1, w2 corresponds to network utilization, w3 corresponds to idle CPU percentage and w4 corresponds to length of jobs in the job queue. The sum of the weighted values is equal to 1. If the weighted value of w3 or w4 is too small, the scheduler may assign the job to a heavily loaded node thereby increasing the job completion time. Network utilization mainly affects the transfer time especially when the job size is large. If the selected node has a lower bandwidth, selecting it will lead to increase in the transfer time. Hence the weighted values are decided as follows:

While computing the performance gain function, the weighted values for the load on a local node is considered as w1=w2=0.1, w3=w4=0.4 and the weighted values for the load on a neighbor node is computed as w1=0.1, w2=w3=w4=0.3. In both the sets of weighted values, the weighted value for memory utilization i.e. w1 is assigned a minimum weight. This is because the technique of memory is more advanced now and the memory utilization may have less influence when compared to that of other attributes [8].

To investigate the fault-tolerance of the ASA and PD_MinRC, I simulate churn and catastrophic failures. Churn is the constant process of node arrival and departure [8]. The failures are modeled as in [8]. In churn failure, every time a node of the group leaves, another node replaces it, so that the network size is maintained. A catastrophic failure is the concurrent failure of randomly large set of nodes. The failure tests are performed with the same parameters as in the simulation tests.

6.2 Performance Metrics

1. Response Time: This metric reflects the ability of an algorithm in minimizing the job completion time.
2. Throughput: This metric represents the number of jobs that are executed per second.
3. Allocation Time: This metric is defined as the time elapsed between the job submitted and it being scheduled. It measures the cost of the fault tolerant scheduling algorithm as perceived by the user.
4. Guarantee Ratio: Total number of jobs guaranteed to meet their deadlines/Total number of jobs * 100% [6].

6.3 Results

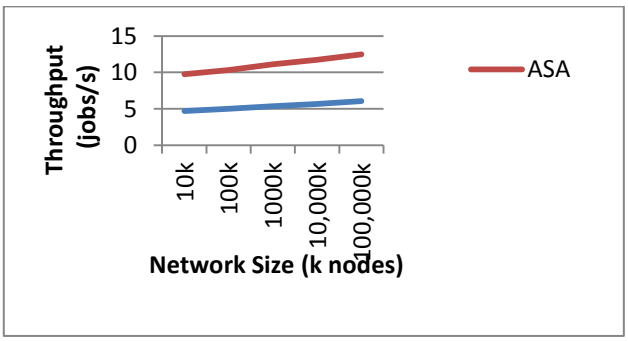

Fig.1. Average Throughput by Network Size

Figure 1 presents the average throughput for varying network size. Parameters other than network size are kept constant. The number of jobs per second submitted to network ranges from 5 to 15. The number of nodes simulated ranges from 10K to 100,000K. I have considered the absolute values for the throughput as both algorithms tries to maximize the throughput. The

throughput in both cases increases linearly with increase in network size. ASA maximizes the throughput by 35.7 percent as compared to PD_MinRC. This is because ASA considers uses the efficiency gain technique as compared to PD_MinRC.

From figure 1, I are confident that ASA would be able to scale to much larger network sizes without affecting the throughput.

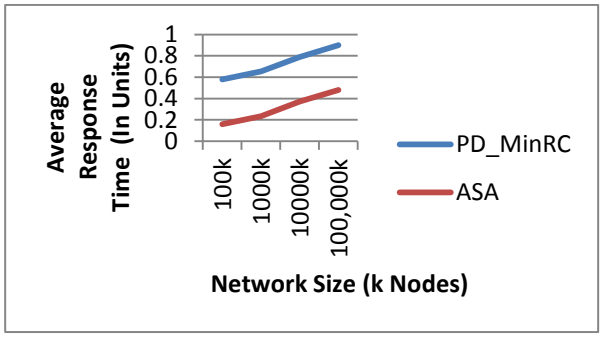

Fig.2. Average Response Time vs. Network Size

In Figure 2, I investigate the average response time of ASA and PD_MinRC under varying network size. The average response time is measured in seconds. The number of jobs simulated for both algorithms are 12 compute intensive jobs. The network is scaled from 100K to 1000, 000K through simulation. The curves of the different approaches show that ASA takes on an average of 41.86 percent less time than PD_MinRC for different network scales. This is because the weighted values for the load attributes are dynamically set and while selecting neighbor nodes network utilization is also taken into consideration. PD_MinRC may select overloaded nodes and may need to reselect other nodes.

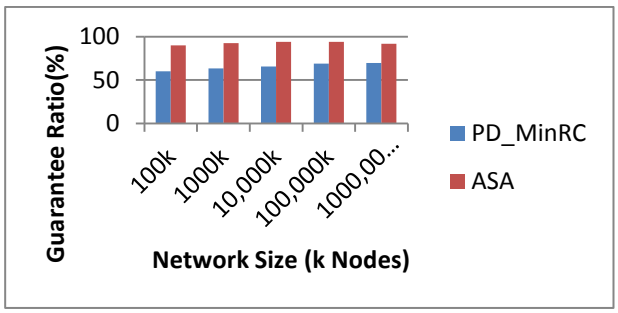

Fig.3. Guarantee Ratio vs. Network Size

The guarantee ratio determines the number of jobs guaranteed to meet their deadlines. The network is scaled from 100k to 1000, 000k. The number of jobs submitted to the network is 12 compute intensive jobs.

Figure 3 shows that with increase in network size, the guarantee ratio of ASA and PD_MinRC gets increased. But the increase in guarantee ratio of ASA is 30.25 percent higher than that of PD_MinRC because of ASA's adaptive nature. This is because ASA considers uses the efficiency gain technique as compared to PD_MinRC.

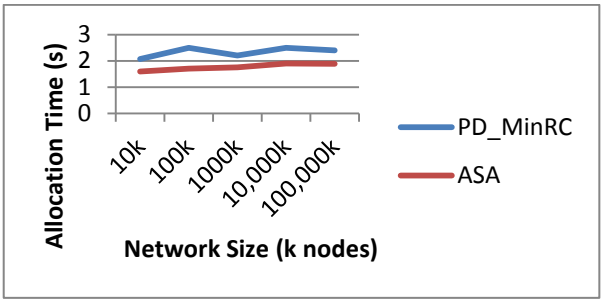

Fig.4(a). Allocation Time vs. Network Size (Slow Link)

Figures 4(a) and 4(b) compare the average allocation time of a 12 jobs. The network size is scaled from 10k to 1000, 000k for the slow link and from 50k to 250k for the fast link. The reason for variation in the network sizes for slow and fast link is to demonstrate the logarithmic increase in allocation time for faster links.

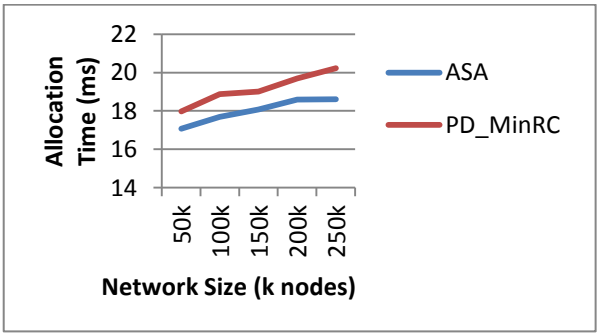

Fig.4(b). Allocation Time vs. Network Size (Fast Link)

Figure 4(a) shows the results for a slow network link model while the 4(b) is plotted with the results of the fast network link model. It is seen that the increase in the allocation time with increase in network size is logarithmic for both ASA and PD_MinRC. PD_MinRC takes an average allocation time of 42.67% more as compared to ASA. From these results, I can highlight that ASA is faster than PD_MinRC on low and high delay interconnection networks.

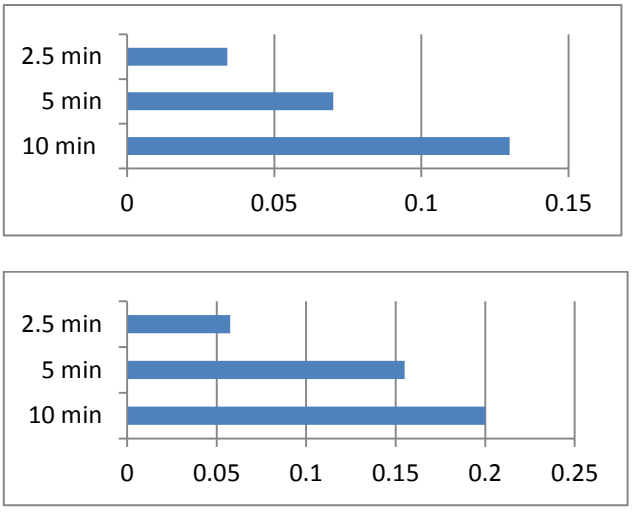

Fig.5. Normalized finished computation for simulations with churn with median session times of 10, 5 and 2.5 min (a) PD_MinRC (b) ASA

Figure 5 shows that ASA is still capable of maintaining its functionality with churn instead of failing and that the performance degradation in ASA is 56.3 percent less as compared to PD_MinRC. In churn failure, every time a node of the group leaves, another node replaces it, so that the network size is maintained. A catastrophic failure is the concurrent failure of randomly large set of nodes. The failure tests are performed with the same parameters as in the simulation tests.

It is seen that churn reduces more the performance of PD_MinRC. This is because deadlines are a heavy requirement and since PD_MinRC employs only passive replication scheme, some of the re-submitted jobs cannot meet their deadlines.

6.4 Comparison With Other Works

Several solutions presented till date do not have the strong substantiation for their scalability and fault-tolerance. The problems that Ire found are that the investigations are performed with only some thousand nodes without considering failures or realistic loads. Table 1 compares our work with that of other works.

Table 1. Comparison to other works

Work Name	Number of Nodes	Failure	LB	Work load
ASA	1,000,000	Yes	Yes	Traces
PD_MinRC [7]	500	Yes	Yes	Poisson Distribution
iHLBA [13]	100	No	Yes	Random Generation
GA [15]	5	No	Yes	Random Generation
QAFT [6]	256	Yes	No	Uniform Distribution
New Model [8]	1,000,000	Yes	No	Traces

More issues where found with other decentralized scheduling platforms [7], [13], [15], [6] and [8]. They experiment with less than 10,000 nodes, no node failures or synthetic workloads.

VII. Conclusion & Future Work

I have presented a decentralized fault-tolerant load balancing model for computing grids called ASA. ASA is the extended study of PD_MinRC. ASA takes into account the communication times; dispatching times, network bandwidth, memory and data storage. It employs backup-copy overlapping technology, striving to advance the start time of primary schedules and delay the start time of backup schedules within the timing constraints. ASA adaptively switches between active and passive replication schemes to reduce the job make-span, optimize node utilization and improve system reliability. ASA scales to a million nodes and tolerates high failure rates. ASA is the first of its kind reported in the literature; it comprehensively addresses the issues of fault tolerance, load balancing, reliability, flexibility and scalability.

To evaluate the performance of ASA, I conduct trace-driven simulations using a million nodes and compared ASA with PD_MinRC. The allocation cost and communication overhead show a logarithmic behavior with increase in the system size, while throughput increases linearly. The other experimental results show that, ASA significantly improves the schedulability of jobs as compared to PD_MinRC for computing grids. When failure occurs, ASA is able to recover its functionality as compared to PD_MinRC 56 percent as churn failure tests and 47.02 percent as catastrophic failure tests report.

In the future, I want to take into consideration other user requirements and test our algorithm on a real computing grid.

References

[1] Minoli, Daniel, "*A Networking Approach to Grid Computing,"* Wiley-Interscience, 2004.

[2] Lu, Kai, Riky Subrata, and Albert Y. Zomaya, "On the Performance-driven Load Distribution for Heterogeneous Computational Grids," *Journal of Computer and System Sciences* vol. 73, no. 8 pp. 1191-1206, 2007.

[3] Ernemann, Carsten, Volker Hamscher, UI Schwiegelshohn, Ramin Yahyapour, and Achim Streit, "On Advantages of Grid Computing for Parallel Job Scheduling," In *Cluster Computing and the Grid, 2nd IEEE/ACM International Symposium on*, pp. 39-39, 2002.

[4] Subrata, Riky, Albert Y. Zomaya, and Bjorn Landfeldt, "Artificial Life Techniques for Load Balancing in Computational Grids," *Journal of Computer and System Sciences* vol.73, no. 8 pp. 1176-1190, 2007.

[5] Erciyes, Kayhan, "A Replication-based Fault Tolerance Protocol Using Group Communication for the Grid," In *Parallel and Distributed Processing and Applications*, Springer Berlin Heidelberg, pp. 672-681, 2006.

[6] Zhu, Xiaomin, Xiao Qin, and Meikang Qiu. "QoS-aware Fault-tolerant Scheduling for Real-time Tasks on Heterogeneous Clusters," *IEEE Transactions on Computers*, vol. 60, no. 6, pp. 800-812, 2011.

[7] Balasangameshwara Jasma and Nedunchezhian Raju, "Performance-Driven Load Balancing with Primary-Backup Approach for Computational Grids with Low Communication Cost and Replication Cost," *IEEE Transactions on Computers*, vol. 62, no. 5, pp. 990-1003, 2013.

[8] Celaya, Javier, and Unai Arronategui, "A Task Routing Approach to Large-Scale Scheduling," *Future Generation Computer Systems*, vol.29, pp. 1097-1111, 2013.

[9] Zheng, Qin, Bharadwaj Veeravalli, and Chen-Khong Tham. "On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs," *IEEE Transactions on Computers*, vol. 58, no. 3, pp. 380-393, 2009.

[10] K. Lu, R. Subrata, and A. Y. Zumaya, "On the Performance Driven Load Distribution for Heterogeneous Computational Grids," Journal of Computer and System Science, vol. 73, no. 8, pp. 1191-1206, 2007.

[11] T. D. Braun, H.J. Siegel and N. Beck, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," Journal. Of Parallel and Distributed Computing, vol. 61, pp. 810-837, 2001.

[12] S.-D. Wang, I.-T. Hsu and Z.-Y. Huang, "Dynamic Scheduling Methods for Computational Grid Environments," Proc. Int'l Conf. Parallel and Distributed Systems, vol. 1, pp.

22-28, 2005.
[13] Y.-H. Lee, S. Leu and R.-S. Chang, "Improving Job Scheduling Algorithms in a Grid Environment," Future Generation Computer Systems, vol. 27, pp. 991-998, 2011.
[14] W. Luo, J. Li, F. Yang, G. Tu, L. Pang and L. Shu, "DYFARS: Boosting Reliability in Fault-Tolerant Heterogeneous Distributed Systems through Dynamic Scheduling," Proc. Eighth ACIS Int'l Conf. Software Eng., Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD '07), pp. 640-645, 2007.
[15] Yajun Li, Yuhang Yang, Maode Ma, Liang Zhou, "A Hybrid Load Balancing Strategy of Sequential Tasks for Grid Computing Environments," Future Generation Computer Systems, no. 25, pp. 819-828, 2009.
[16] Khan, Rafiqul Z., and Md F. Ali. "An Efficient Diffusion Load Balancing Algorithm in Distributed System." International Journal of Information Technology and Computer Science (IJITCS) 6, no. 8 (2014): 65.
[17] Rahdari, Danial, Amir Masoud Rahmani, Niusha Aboutaleby, and Ali Sheidaei Karambasti. "A Distributed Fault Tolerance Global Coordinator Election Algorithm in Unreliable High Traffic Distributed Systems." International Journal of Information Technology and Computer Science (IJITCS) 7, no. 3 (2015): 1.

Authors' Profiles

Jasma Balasangameshwara received the BE and MTech degrees in Information Technology & Engineering and Computer Science& Engineering from Visveshwaraiah Technological University, India, in 2005 and 2008, respectively. She received her Ph.D. degree in Information & Communication Technology from Anna University, India in 2013. She is currently working as Associate Professor at the Atria Institute of Technology, Bangalore. Her research interests include cluster computing, fault-tolerant computing and distributed computing.

How to cite this paper: Jasma Balasangameshwara,"Improving Fault-Tolerant Load Balancing Algorithms in Computational Grids", IJIEEB, vol.7, no.6, pp.53-62, 2015. DOI: 10.5815/ijieeb.2015.06.08