Availability Prediction Based Replication Strategies for Grid Environments

Brent Rood  
Department of Computer Science  
State University of New York at Binghamton  
Binghamton, NY, USA  
brood1@binghamton.edu

Michael J. Lewis  
Department of Computer Science  
State University of New York at Binghamton  
Binghamton, NY, USA  
mlewis@cs.binghamton.edu

Abstract

Volunteer-based grid computing resources are characteristically volatile and frequently become unavailable due to the autonomy that owners maintain over them. This resource volatility has significant influence on the applications the resources host. Availability predictors can forecast unavailability, and can provide schedulers with information about reliability, which helps them make better scheduling decisions when combined with information about speed and load. This paper studies using this prediction information for deciding when to replicate jobs. In particular, our predictors forecast the probability that a job will complete uninterrupted, and our schedulers replicate those jobs that are least likely to do so. Our strategies outperform other comparable replication strategies, as measured by improved makespan and fewer redundant operations. We define a new “replication efficiency” metric, and demonstrate that our availability predictor can provide information that allows our schedulers to be more efficient than the most closely related replication strategy for a variety of loads in a trace-based grid simulation. We demonstrate that under low load conditions, our techniques come within 6% of the makespan improvement of a previously proposed replication technique while creating 76.8% fewer replicas and under higher loads, can improve makespan marginally while creating 72.5% fewer replicas.

1. Introduction

Grids that comprise large collections of individual machines, as opposed to (or in addition to) the dedicated clusters that are characteristic of institutional grids, will challenge schedulers and middleware to effectively navigate around failures and unavailability [1][2]. In particular, resources that are not dedicated to grid applications, and that are also used for local jobs, will vary as to when and to what extent they can be used by the grid. This will depend on local application characteristics, resource owner usage patterns, the day of the week, the time of day, the type of machine (e.g. laptops may have more intermittent availability than servers), and other factors [3][4]. It will also depend on the sharing policy enacted by the resource owner and enforced by the grid middleware.

Availability models [5][3] capture this behavior, and can help characterize the availability behavior of each machine. Moreover, resource availability predictors can use past (and potentially group) behavior to help forecast whether a resource will become unavailable, and consequently, the probability that a job will run to completion [6]. If a machine fails while executing a job, the type of job will determine the effect of the failure. Jobs that cannot take checkpoints may have to start over. Jobs that can take on-demand checkpoints may tolerate brief periods of unavailability (but not hard failures). Those that take periodic checkpoints may be best equipped to deal with unavailability, but must pay the cost of potentially unnecessary checkpoints.

Previously, we have shown that schedulers that incorporate our predictors’ estimates of future reliability, along with load and processor speed, can decrease average makespan and waste fewer operations by making better initial job placement decisions. Scheduling non-checkpointable jobs differently from checkpointable jobs can help overall grid utilization (by saving the reliable resources for jobs that do not checkpoint themselves) [7].

In this paper, we investigate the efficacy of using our availability predictors in a completely new way, namely to decide which jobs to replicate. The basis for this idea is that checkpointing and replication are two of the primary tools for dealing with resource unavailability. The system cannot generally add checkpointability; the application programmer must do that. So whereas schedulers can (and should) take advantage of knowing which jobs are checkpointable (an idea we have explored previously [7]), they cannot proactively add reliability by increasing the checkpointability of the job mix.

The system can, however, replicate some jobs in an attempt to deal with possible resource unavailability. Replicating a job can benefit performance in one of two ways. First, jobs are scheduled onto resources using imperfect ranking metrics that may or may not reflect how fast a job will run on a machine. Therefore, by starting a job to run simultaneously on more than one resource, the application makespan is determined by the earliest completion time among all replicas. As this can depend on unpredictable load and usage patterns, the replica can potentially improve the makespan of the job. Second, replicated job executions can also help deal with failure; then when one resource fails, the adverse effect on performance of the jobs it runs can be reduced if replicas complete without failure.

Replication does not come without a cost, however. Within a grid economy, it is likely that applications will need to pay for grid use on a per job, per process, or per operation basis. Therefore, replication can cost extra, assuming redundant jobs are counted separately (which seems likely). Replication can also have an adverse indirect effect, as some of the highest
ranked resources could be used for executing replicas, leaving only “worse” (by whatever metric the scheduler uses to rank jobs) resources for subsequent jobs.

We therefore set out to test the hypothesis that our availability predictor can help select the right jobs to replicate, and can therefore improve overall average job makespan, reduce the redundant operations needed for the same improved makespan, or both. We explore replication strategies that make copies of all jobs, some fixed percentage of jobs, long jobs, jobs that are not checkpointable, and several combinations. We test these strategies under a variety of loads using two availability traces [8][3]. We then explore the effectiveness of replicating only the jobs that are mapped onto resources that fall below some reliability threshold. That is, we ask the predictor to forecast the likelihood that a resource will complete a particular job without becoming unavailable. If this predicted probability value is too low, we replicate the job. If the resource is deemed reliable enough, the job is not replicated.

This paper makes four primary contributions. First, Section 5 describes a new metric, replication efficiency, to analyze replication effectiveness. Second, Section 5.1 analyzes a replication strategy for using availability predictions to determine when to replicate jobs. We investigate how this replication strategy affects both makespan and replication efficiency. Third, Section 5.2 describes how system load effects replication effectiveness. Finally, Section 5.3 proposes three different load adaptive replication techniques, which consider the system load and the desired metric to decide which replication strategy to use.

2. Background

In previous work, we have investigated several approaches to predicting when resources will transition to unavailability. Each approach analyzes some period of a resource’s recent history to forecast future availability. Details of our prediction policies are provided elsewhere [6]. Our predictors take a resource, prediction time, and estimated application duration (i.e. “prediction interval”) as input, and generate as output a percentage between 0% and 100% that represents the likelihood that the resource will complete the interval without becoming unavailable; a value of 100% indicates the highest level of certainty that the resource will remain available. It is this value that we use to drive the replica creation policies for the schedulers in this paper. That is, we use our best predictor (TRF [7]) to output the probability that a resource will remain available for the full runtime of the application; a predicted probability value below some configurable Replication Score Threshold triggers the creation of a replica.

3. Related Work

The novelty of this paper is based on using availability predictions that feed into schedulers to enable resource reliability and application characteristics to be considered when replicating grid jobs. These components lead to a new strategy for improving makespan and reducing overhead through judicious job replica creation.

In availability prediction, Ren et al. [5] [9] [10] use empirical host CPU utilization and resource contention traces to develop an availability model, prediction technique, and prediction-based scheduler. Importantly and as described in our previous work, the differences in our prediction techniques lead to our predictors producing improved prediction accuracies. Our prediction techniques analyze different portions of a resource’s history to enable predictions and calculate those predictions with an improved technique (including a transition weighting scheme) which has been shown to produce higher prediction accuracies [6]. Previous papers [3][7] have described in detail the related work in the areas of availability analysis (most notably Ren et. al. [5]), load and availability prediction (including NWS [11], RPS [12], and others [10][13][14][11][15][16]), and reliability-based scheduling [17][18]. We therefore focus the rest of this section on job replica creation and placement in grid scheduling.

Replication and checkpoint-restart are widely studied techniques for improving fault tolerance and performance. Data replication makes and distributes copies of files in distributed file sharing systems or data grids [19][20]. These techniques strive to give users and applications more efficient access to data by moving it closer, and to mitigate the effects of failure. Other work considers replica location when scheduling tasks. Santos-Neto et al. [21] schedule data-intensive applications, and introduce Storage Affinity, a heuristic scheduling algorithm that exploits a data reuse pattern to consider data transfer costs and ultimately reduce application makespan.

Task replication makes copies of jobs, again for both fault tolerance and performance. Li et al. [22] strive to increase throughput and decrease grid application execution time, by determining the optimal number of task replicas for a simulated and dynamic resource environment. Their analytical model determines the minimum number of replicas needed to achieve a certain task completion probability at a specified time. They compare dynamic rescheduling with replication, and extend the replication technique to include a N-out-of-M scheduling strategy for Monte Carlo applications. Similarly, Litke et al. [23] present a task replication scheme for a mobile grid environment. They model resources according to a Weibull reliability function, and estimate the number of task replicas needed for certain levels of fault tolerance. The authors use a knapsack formulation for scheduling, to maximize system utilization and profit, and evaluate their approach through simulation.

Silva et al. [24] investigate scheduling independent tasks in a heterogeneous computational grid environment, without host speed, load, and job size information; the authors use replication to cope with dynamic resource unavailability. Workqueue with Replication (WQR) first schedules all incoming jobs, then uses the remaining resources for replicas. The authors use simulation to compare WQR with various maximum amounts of replicas (1x, 2x, 3x, etc), to Dynamic FPLTF [25] and
Sufferage [26] through simulation. Angalano et al. [27] later extend this work with a technique called WQR Fault Tolerant (WQR-FT), which adds checkpointing to the algorithm. Since in WQR a failed task is abandoned and never restarted, WQR-FT adds automatic task restart to keep the number of replicas of each task above a certain threshold. Tasks may use periodic checkpoints upon restart. In Section 5, we use WQR-FT for comparison against our strategies.

Others investigate the relationship between checkpoint-restart and replication. Weissman [28] develops performance models for quantitatively comparing the separate use of the two techniques in a grid environment. Similarly, Ramakrishnan et al. [29] compare checkpoint-restart and task replication by first analytically determining the costs of each strategy and then providing a framework that enables plug and play of resource behavior to study the effects of each fault tolerant technique under various parameters.

Importantly, none of the related work uses on demand individual resource availability prediction or application characteristics such as length and checkpointability to determine if a task replica should be created. We use a mix of checkpointable and non-checkpointable jobs, and drive our simulations with real world resource traces that capture resource unavailability. We also introduce a new metric for studying the efficiency of replication strategies and improve upon results obtained by existing techniques.

4. Experimental Setup

Our results are based on two traces, a six month Notre Dame Condor availability trace of 700 nodes [3] and a trace from the SETI@home desktop grid system [8].

For the Condor simulations, the MIPS score, the load on each processor, and the (un)availability states (included or implied by the trace data) all influence the simulated running of the applications. Resources are considered available if they are running, connected, have no user present and have a local CPU load below 30% as these are the default settings for the Condor system [30]. A resource may only be assigned one task for execution at a time. Similarly, for the SETI simulations the double precision floating point speed and the (un)availability states (implied by the trace data) influence the execution of the applications. In the SETI trace, a machine is considered available if it is eligible for executing a grid application. This is in accordance with the resource owner’s settings which may dictate that a machine is only available if the CPU is idle and the user is not present. Since the SETI trace contains 226,208 resources over 1.5 years, we randomly choose 700 resources and only execute the simulation on the first 6 months of the trace to compare directly to the results obtained from the Condor trace simulations.

We simulate executing applications by inserting them at random times throughout the six months, with durations uniformly distributed between five minutes and 24 hours. 1 The duration is determined by the runtime on an unloaded resource with average speed.

A uniform distribution demonstrates how each replication technique performs for a wide range of job lengths, without emphasizing a particular set of lengths. 25% of the applications are checkpointable, and can take an on-demand checkpoint in five minutes, if the user reclaims a resource, or if the local CPU load exceeds 50%. If the resource directly transitions to an unavailable state (for example, by failing), no checkpoint can be taken and the job must restart from its last periodic checkpoint. 2 Non-checkpointable applications must restart from the beginning of their execution when evicted regardless of the eviction type. In both cases, an evicted job is added to the job queue and immediately rescheduled on an available resource.

Jobs are scheduled each round (once every 3 minutes) by removing and scheduling the job at the head of the queue until a job can no longer be scheduled. For each job, each available resource is scored and the resource with the highest score executes the job. Resource i’s score (RS(i)) is computed as:

\[
RS(i) = m_i \cdot (1 - l_i)
\]

where \(m_i\) is i’s MIPS score, and \(l_i\) is the CPU load currently on resource \(i\).

When a job is first scheduled, the scheduler determines if a single replica of the task should be created based on the current replication policy. 3 If the scheduler chooses to create a replica, an identical copy of the job is then scheduled. This replica is not considered for further replication. When any replica of a task completes, all other replicas of the task are located and terminated, freeing up the resources executing those replicas.

Some replication policies use an availability prediction. For this we use our TRF predictor described in previous work [6]. TRF takes as input the expected duration of the job and the name of a resource, and returns a percentage from 0 to 100 to represent the predicted probability that the resource will complete the expected duration without becoming unavailable.

In our experiments, we vary the total number of jobs in the system, using 1K, 14K, 27K, and 40K total jobs over the 6 month traces, in four separate sets of simulations. This translates to 0.001, 0.13, 0.26, and 0.39 Jobs per Resource per Day, respectively. These same values were used for the rest of the tests described in this paper, we refer to them as the Low, Medium Low, Medium High, and High load cases. We analyze scheduling quality according to average job makespan, and replication efficiency (which is defined and discussed later). We calculate makespan for each job as the time from submission to completion; and average this value across all jobs. Task lengths are defined in terms of the number of operations needed to complete them.

2. Periodic checkpoints are taken every 6 hours (Condor’s default).

3. Although we’ve obtained algorithms and results for an increased number of replicas per job, we restrict this study by limiting the number of replicas to one per job due to space constraints.
5. Prediction-based Replication

Intuitively, replicating non-checkpointable jobs yields larger rewards than replicating checkpointable jobs due to the inherent fault tolerance of checkpointable jobs. Our experiments have confirmed this intuition; replicating only non-checkpointable jobs improves makespan and replication effectiveness. Replicating longer jobs also yields higher improvements.

We improve on these replication strategies by indentifying and replicating the jobs that are most likely to fail. For this work, we select a resource based on its projected near-future performance (speed and load) as described in Section 4, and replicate based on the probability of completing the job on that resource. We replicate the jobs that are more likely to fail. Additionally, we propose a method for considering the makespan improvement and overhead within the same metric, to quantify how much of a makespan improvement each replication policy can achieve per replica created. We define Replication Efficiency as:

\[
\text{Replication Efficiency} = \frac{\text{Makespan Improvement}}{\text{Replicas Per Job}}
\]

where Makespan Improvement is the percentage makespan improvement over not creating any replicas. As defined, increasing makespan improvement with the same number of replicas will increase efficiency, and increasing replicas to get the same makespan improvement will decrease efficiency.

Another way of considering efficiency is to see how much makespan improvement can be achieved with a certain number of replicas. The best replication strategies will make replicas of the “right” jobs, and achieve more improvement for the same cost (number of replicas).

Replication efficiency is important in any environment that applies a cost to inserting and executing jobs on a system. One user may wish to complete a job as soon as possible (reduce makespan) regardless of cost, whereas a frugal user may only wish to replicate if the replica created is likely to provide significant makespan improvement; the frugal user may choose an extremely efficient replication policy. Efficiency can also be useful to administrators in choosing which replication policy is most suited to the goals of their system. Efficient replication strategies reduce the overhead and increase the throughput of the overall system by reducing the number of wasted operations performed by replicas that do not complete, that fail, or that are evicted. This section analyzes methods for using the availability predictions to make better replication decisions in terms of both makespan improvement and efficiency.

5.1. Replication Score Threshold

This section describes how the prediction threshold at which the system makes a replica affects performance. The Replication Score Threshold (RST) is the predicted completion probability, as calculated by TRF, below which the system makes a replica, and above which it does not. We study how RST affects both makespan improvement and replication efficiency. For example, a system with an RST of 100 replicates all jobs scheduled on resources with a predicted completion probability below 100%. Additionally, we propose RST-L-NC which chooses jobs to replicate based their predicted completion probability (RST), their Length (longer jobs are replicated), and whether they are checkpointable (only Non-Checkpointable jobs are replicated). We compare these results with a policy (1x) that replicates each job, and a policy that replicates on available resources only (WQR-FT) [27].

To summarize,

- **1x**: Creates a single replica of each job
- **RST**: Given an availability prediction from the TRF predictor, replicates all jobs whose predicted completion probability falls below the configured RST value
- **RST-L-NC**: Replicates all non-checkpointable (NC) jobs that are longer than ten hours (L), and given an availability prediction from the TRF predictor, whose predicted completion probability falls below the configured RST value
- **WQR-FT**: First schedules all incoming jobs, then uses the remaining resources for replicas by creating up to one replica of each job until either all jobs have a single replica executing or there are no longer any available resources

Figure 1 plots makespan and efficiency versus RST in the “Low” load case. The top two graphs represent the results from executing the simulation on the Condor trace and the bottom two graphs present the results from the SETI trace.

As RST increases, more jobs are replicated as more predicted completion probabilities fall below the RST. In the low load case, this increases makespan improvement for both RST and RST-L-NC. The effect is more prominent in the Condor simulations, but still evident in the SETI simulations. 1x and WQR-FT achieve the highest makespan improvement in the Condor simulations but are matched by RST in SETI.

The efficiency graphs exhibit the opposite trend. For all RST values, both RST and RST-L-NC produce higher efficiencies than WQR-FT and 1x. In fact, RST-L-NC produces a 39.5 (Condor) and a 47.5 (SETI) replication efficiency versus WQR-FT’s efficiency of 21 (for both SETI and Condor). RST-L-NC’s efficiency outperforms RST due to the incorporation of job characteristics in choosing to replicate longer, non-checkpointable jobs. The more intelligently and selectively the scheduler chooses which jobs to replicate, the higher the achieved efficiency. Clearly, considering both predicted completion probability and an application’s characteristics yields the most efficient strategy. For example in SETI, RST matches the makespan improvement of WQR-FT but creates 35% fewer replicas while RST-L-NC comes within 6% of the makespan improvement of WQR-FT while creating 67% fewer replicas. Similarly in Condor, RST-L-NC comes within 6% of the makespan improvement of WQR-FT but creates 76.8% fewer replicas.

4. An RST of 100 will not replicate all jobs but only jobs whose predicted probability of completion is below 100%. If a resource has not yet failed within the period of time being traced, the predictor may output a 100% probability of completion.
5.2. Load Adaptive Motivation

Next we examine how increasing load on the system influences the replication policies’ effectiveness. Figure 2 plots makespan and efficiency versus RST in the “Medium High” load case. Again, the top two graphs represent the results from the Condor simulations and the bottom two graphs represent the results from the SETI simulations. Under Medium High load, increasing RST actually decreases makespan improvement; creating additional replicas in higher load situations denies other jobs access to attractive resources. RST70 (RST\(i\), where \(i\) is a value between 0 and 100 replicates all jobs whose predicted probability of completion is below \(i\)\%), for example produces the largest makespan improvement for Condor and RST40 produces the largest makespan improvement for SETI. RST and RST-L-NC produce much higher efficiencies than either 1x or WQR-FT across all RST values in both the Condor and SETI simulations. At this load level, RST-L-NC produces a 30 (Condor) and a 20 (SETI) replication efficiency versus WQR-FT’s 7 (Condor) and 8.5 (SETI), respectively. In SETI, RST-L-NC comes within 0.5% of the makespan improvement of WQR-FT while creating 58% as many replicas and RST improves on the makespan by 2.5% and creates 13.7% fewer replicas. For Condor, RST-L-NC improves on the makespan of WQR-FT by 1.5% while creating 72.5% fewer replicas. Again, RST-L-NC produces higher replication efficiency than RST due to its higher replica creation selectivity. Figure 2 demonstrates that under a higher load, selective replication strategies produce larger makespan improvements. Even under low load (Figure 1), these more selective and efficient replication techniques result in more improvement per replica.

Figure 3 further investigates the effect that varying the load in the system has on the performance achieved by the replication techniques proposed in Section 5.1. Figure 3 plots our RST replication technique with RST values of both 100% and 40% and compares them with not replicating (No Replication) and replicating a random 20% of jobs (20% Replication) only for the Condor case due to space constraints (although the result holds for the SETI case). As load increases, the makespan improvement and efficiency of all replication strategies initially increases slightly but then falls as we reach Med-High and High load levels. Intuitively, this is because under lower loads replicas will not interfere by taking quality resources away from other jobs. As load increases, replicas can cause new jobs to wait. Another important result is comparing our techniques to blind replication. For example, the RST40-L-NC technique creates almost exactly the same number of replicas as randomly replicating 20% of jobs across the loads.
levels. Given the same number of replicas, the RST technique achieves a higher makespan improvement and efficiency across all load levels; it is choosing the “right” jobs to replicate.

Figure 3 also shows that the replication strategy that provides the highest achievable makespan or efficiency varies based on the load of the system. For example, depending on the load level, either $\text{RST}_{40}\text{-L-NC}$ or $\text{RST}_{100}\text{-L-NC}$ produces the largest makespan improvement.

5.3. Load Adaptive Techniques

Since load varies unpredictably, the replication policy should adapt dynamically. And because different users may target different metrics, the replication policies must also vary by job, suggesting three load adaptive replication techniques that respond to varying load levels by choosing the replication strategy that best achieves three different desired metrics. A load measurement system counts the number of jobs submitted to the system in the past day. This value then determines the current load level and helps select a replication strategy. We define Low load as fewer than 40 jobs per day, Med-Low load as fewer than 115 jobs per day, Med-High load as fewer than 200 jobs per day and High load as 200 or more jobs a day. RST-based strategies consider different combinations of checkpointability and job length depending on the current system load and desired metric.

- **Performance**: Reduce average job makespan across loads
  - Low load: 4x - Create 4 replicas of each job
  - Med-Low load: RST$_{100}$ - Replicate a job if the completion probability is less than 100%
  - Med-High load: RST$_{40}$-NC - Replicate a job if the completion probability is less than 40% and the job is non-checkpointable
  - High load: No replication

- **Efficiency**: Increase replication efficiency across loads
  - Low, Med-Low, Med-High: RST$_{40}$-L-NC - Replicate a job if the completion probability is less than 40%, the job is non-checkpointable and the length is over 10 hours
  - High load: No replication

- **Compromise**: Compromise between average job makespan and efficiency across loads
  - Low load: RST$_{100}$ - Replicate a job if the completion probability is less than 100%
  - Med-Low load: RST$_{100}$ - Replicate a job if the completion probability is less than 100%
  - Med-High load: RST$_{40}$-NC - Replicate a job if the completion probability is less than 40% and the job is non-checkpointable

Fig. 2. Replication Score Threshold’s effect on replication effectiveness - Medium High Load
In Figure 4, we explore the results of our three proposed load adaptive replication policies across varying load levels and compare their results to the closest related work replication strategy, WQR-FT. Again, the top two graphs represent the results from the Condor simulations and the bottom two illustrate the SETI results. For both cases, all of our load adaptive techniques produce the largest (or extremely close to the largest) achievable value for their chosen metric at each given load level with the exception of the high load case for SETI. The Efficiency technique produces the largest efficiency across all load levels (except in the high load case in SETI) while simultaneously producing the smallest makespan improvement. The Performance technique produces the opposite trend. The Compromise technique produces an intermediate makespan improvement and efficiency across all loads. The results demonstrate these replication techniques can determine the replication strategy that is best suited to achieving the desired metric at various loads.

Compared to WQR-FT, all three load adaptive techniques produce higher replication efficiencies across all loads except for the Performance technique in the low load case and the high load case in SETI. The Efficiency technique produces an average replication efficiency increase of 10.8 across all loads with a maximum increase of 22 compared with WQR-FT. Similarly, the Efficiency technique produces at most an increase of 25 in efficiency for the Condor simulations and on average increase of 15 across all loads compared with WQR-FT. The Performance technique beats the makespan improvement achieved by WQR-FT by an average of 1.27% across all loads in the Condor simulations while simultaneously achieving a higher efficiency in all but the low load case. The Performance technique also improves on the makespan improvement of WQR-FT in all but the high load case in SETI. The Compromise technique improves or matches the efficiency and makespan achieved by WQR-FT across all loads with the exception of the high load case in the SETI simulations.

6. Conclusion

Scheduling in a large scale grid that comprises a heterogeneous collection of eclectic resources requires techniques for dealing with resource failure and unavailability. Checkpointing and replication are the two primary tools that grids can use to mitigate the effect of this resource volatility. Schedulers can replicate jobs to reduce the effect of failure and ultimately reduce makespan. However, replication requires additional grid cycles, which can have direct cost within a grid economy, and an indirect cost in tying up attractive resources especially under relatively higher loads. This paper describes techniques that use availability predictions to influence a scheduler’s decision about whether to replicate a job. Availability predictors can forecast resource behavior, and allow schedulers...
to consider resource reliability when making job replication decisions. We introduce the notion of “replication efficiency”, which combines the benefit and cost of replication into a single metric, and study a variety of replication policies under various loads, using simulations based on two representative grid resource traces to demonstrate the improved performance our predictor and replication techniques produce. A strategy that considers process checkpointability, job length, predicted resource reliability and current system load does the best job of using additional replicated operations to reduce job makespan efficiently.

References