J Grid Computing (2009) 7:479–500
DOI 10.1007/s10723-009-9135-2

Grid Resource Availability Prediction-Based Scheduling and Task Replication

Brent Rood · Michael J. Lewis

Received: 20 February 2009 / Accepted: 18 August 2009 / Published online: 2 September 2009

Abstract The frequent and volatile unavailability of volunteer-based Grid computing resources challenges Grid schedulers to make effective job placements. The manner in which host resources become unavailable will have different effects on different jobs, depending on their runtime and their ability to be checkpointed or replicated. A multi-state availability model can help improve scheduling performance by capturing the various ways a resource may be available or unavailable to the Grid. This paper uses a multi-state model and analyzes a machine availability trace in terms of that model. Several prediction techniques then forecast resource transitions into the model's states. We analyze the accuracy of our predictors, which outperform existing approaches. We also propose and study several classes of schedulers that utilize the predictions, and a method for combining scheduling factors. We characterize the inherent tradeoff between job makespan and the number of evictions due to failure, and demonstrate how our schedulers can navigate this tradeoff under various scenarios. Lastly, we propose job replication techniques, which our schedulers utilize to replicate those jobs that are most likely to fail. Our replication strategies outperform others, as measured by improved makespan and fewer redundant operations. In particular, we define a new metric for replication efficiency, and demonstrate that our multi-state availability predictor can provide information that allows our schedulers to be more efficient than others that blindly replicate all jobs or some static percentage of jobs.

Keywords Multi-state · Prediction · Availability · Characterization · Scheduling · Replication

This research is supported by NSF Award CNS-0454298.

B. Rood (✉) · M. J. Lewis
Department of Computer Science,
State University of New York at Binghamton,
P.O. Box 6000 Binghamton, NY 13902-6000, USA
e-mail: brood1@binghamton.edu

M. J. Lewis
e-mail: mlewis@cs.binghamton.edu

1 Introduction

The functionality, composition, utilization, and size of large scale distributed systems continue to evolve. The largest Grids and testbeds under centralized administrative control—including TeraGrid [47], EGEE [14], Open Science Grid (OSG) [18], and PlanetLab [33]—vary considerably in terms of the number of sites and the

extent of the resources at each. Various peer-to-peer (P2P) [4] and Public Resource Computing (PRC) [3] systems allow users to connect and donate their individual machines, and to administer their systems themselves. We expect hybrids of these various Grid models and middleware to emerge as well [1, 15]. Future Grids will contain dedicated high performance clusters, individual less-powerful machines, and even a variety of alternative devices such as PDAs, sensors, and instruments. They will run Grid middleware, such as Condor [27] and Globus [16], that enables a wide range of policies for contributing resources to the Grid.

This continued increase in functional heterogeneity will make Grid scheduling—mapping jobs onto available constituent Grid resources—even more challenging, partly because different resources will exhibit varying unavailability characteristics. For example, laptops may typically be turned on and off more frequently, and may join and leave the network more often. Individual workstation and desktop owners may turn their machines off at night, or shut down and restart to install new software more often than a server's system administrator might. Even the same kinds of resources will exhibit different availability characteristics. CS department research clusters may be readily available to the Grid, whereas a small cluster from a physics department may not allow remote Grid jobs to execute unless the cluster is otherwise idle. Site autonomy has long been recognized to be an important Grid attribute [24]. In the same vein, even owners of individual resources will exhibit different usage patterns and implement different policies for how available they make their machines.

If Grid schedulers do not account for these important differences, they will make poor mapping decisions that will undermine their effectiveness. Schedulers that know when, why, and how the resources fail, can be much more effective, especially if this knowledge is coupled with information about job characteristics. For example, long-running jobs that do not implement checkpointing may require highly available and reliable host machines. Checkpointable jobs that require heavyweight checkpoints may prefer resources that are reclaimed by users, rather than those that fail, thereby making possible an on demand checkpoint. This would be less important for jobs that produce lightweight checkpoints. Easily replicable processes without side effects, might do well on the less powerful and more intermittently available resources, leaving the other machines for jobs that really need them (at least under moderate to high contention for Grid resources).

Therefore, schedulers must effectively predict the availability of constituent resources. Ignoring resource reliability characteristics can lead to longer job makespans due to wasted operations [13]. Even more directly, it can adversely affect job reliability by favoring faster but less reliable resources that cannot complete jobs before failing or being reclaimed from the Grid by their owners. Unfortunately, performance and reliability vary inversely [13]; favoring one necessarily undermines the other.

This paper therefore relies on the central motivating tenet that Grid schedulers must consider both reliability and performance in making scheduling and replication decisions. We investigate the effect on both job makespan and the number of wasted operations due to evictions, which can occur when a resource executing a job becomes unavailable. Evictions and wasted operations are important because of their direct "cost" within a Grid economy, or simply because they essentially deny the use of the resource by another local or Grid job. Our approach to Grid scheduling involves analyzing resource availability history and predicting future resource (un)availability, monitoring and considering current load, storing static resource capability information, and considering all of these factors when placing jobs.

Sections 3 and 4 describe multi-state availability model and briefly examine a resource availability trace from the University of Notre Dame in terms of that model to establish the volatility of the underlying resources. Section 5 examines several availability prediction techniques used to forecast the future states of a resource's availability. Section 6 investigates using different approaches to schedule jobs with different characteristics based on those predictions. Section 7 then explores the efficacy of using our availability predictors in a new way; we examine how helpful they can be in deciding which jobs to replicate.

2 Related Work

Related work resides in three broad categories, namely (i) prediction of resource state (availability and load), (ii) Grid scheduling, especially approaches that consider reliability and availability, and (iii) job replication. We organize this section accordingly.

2.1 Prediction

The most notable prediction system for Grids and networked computers is Network Weather Service (NWS) [51]. NWS uses a large set of linear models for host load prediction, and combines them in a mixture-of-experts approach that chooses the best performing predictor. The RPS toolkit [12] uses a set of linear host load predictors, including BM(p) (or Sliding Window), which records and averages the states occupied in the last N intervals.

In availability prediction, Ren et al. [38–40] use empirical host CPU utilization and resource contention traces to develop the only other multi-state model, prediction technique, and multi-state prediction based scheduler for resource availability of which we are aware. Their multi-state availability model includes five states, three of which are based on the CPU load level (which resides in one of three zones); the two other states indicate memory thrashing and resource unavailability. For prediction, the authors count the transitions from available to each state during the previous N days to produce a Markov chain for state transition predictions. These transition counts determine the probability of transitioning to each state from the available state.

Mickens and Noble [29–31] use variations and combinations of saturating counters and linear predictors, including a hybrid approach similar to NWS's mixture-of-experts, to predict the likelihood of a host being available for various look ahead periods. A Saturating Counter predictor increases a resource-specific counter during periods of availability, and decreases it during unavailability. A History Counter predictor gives each resource 2^N saturating counters, one for each of the possible availability histories dictated by the last N availability sampling periods. Predictions are made by consulting the applicable counter value associated with the availability exhibited in the last N sampling periods.

Pietrobon and Orlando [34] use regressive analysis of past job executions to predict whether a job will succeed. Nurmi et al. [32] model machine availability using Condor traces and an Internet host availability dataset, attempt to fit Weibull, hyper-exponential, and Pareto models to the availability duration data, and evaluate them in terms of "goodness-of-fit" tests. They then provide confidence interval predictions for availability durations based on model-fitting [10]. Similarly, Kang and Grimshaw filter periodic failures out of resource availability traces and then apply statistical models to the remaining availability data [19]. Finally, several machine learning techniques use categorical time-series data to predict rare target events by mining event sets that frequently precede them [43, 48, 49].

2.2 Scheduling

Most Grid scheduling research attempts to decrease job makespan or to increase throughput. For example, Kondo et al. [21] explore scheduling in a volunteer computing environment. Braun et al. [7] explore eleven heuristics for mapping tasks onto a heterogeneous system, including min-min, max-min, genetic algorithms and opportunistic load balancing. Cardinale and Casanova [8] use queue length feedback to schedule divisible load jobs with minimal job turnaround time. GrADS [11] uses performance predictions from NWS, along with a resource speed metric, to help reduce execution time.

Fewer projects focus on scheduling for reliability. Kartik and Murphy [20] calculate the optimal set of processor assignments based on expected node failure rates, to maximize the chance of task completion. Qin et al. [35] investigate a greedy approach for scheduling task graphs onto a heterogeneous system to reduce reliability cost and to maximize the chance of completion without failure. Similarly, Srinivasan and Jha [46] use a greedy approach to maximize reliability when scheduling task graphs onto a distributed system.

Unfortunately, scheduling only for reliability undermines makespan, and scheduling only on the

fastest or least loaded machines can be detrimental due to the performance ramifications of failures. Dogan and Ozguner [13] develop a greedy scheduling approach that ranks resources in terms of execution speed and failure rate, weighing performance and reliability in different ways. Their work does not use failure predictions, and assumes that each synthetic resource follows a Poisson failure probability with no load variation, and that failed machines never restart. We remove all of these assumptions in our work, and utilize availability and load traces from real resources. Amin et al. [2] use an objective function to maximize reliability while still meeting a real time deadline. They search a scheduling table for a set of homogenous non-dedicated processors to execute tandem real-time tasks. These authors also assume a constant failure rate per processor.

Some distributed scheduling techniques use availability prediction to allocate tasks. Kondo et al. [22] examine behavior on the previous weekday to improve the chances of picking a host that will remain available long enough to complete a task's operations. Ren et al. [39] also examine scheduling jobs with their Ren N-day predictor. The Ren MTTF scheduler first calculates each resource's mean time to failure ($MTTF_i$) by adding the probabilities of exiting available for time t, as t goes from 0 to infinity. It then calculates resource i's effective task length (ETL_i) as:

$$ETL_i = MTTF_i \cdot CR_i \cdot (1 - L_i)$$

CR_i is resource i's clock rate and L_i is its predicted average CPU load. It then selects the resource with the smallest ETL from the resources with MTTF values that are larger than the ETL. If no such resources exist, it selects the resource with the minimum job completion time, considering failures. Our work is most similar to Ren et al.'s, and differs in the following ways: (i) we consider how and why a resource may become unavailable, and attempt to exploit varying consequences of different kinds of unavailability, (ii) we schedule checkpointable and non-checkpointable jobs differently, to improve overall performance, and (iii) we explicitly analyze and schedule for the tradeoff between performance and reliability.

2.3 Replication

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 [23, 37]. These techniques strive to give users and jobs more efficient access to data by moving it closer, and to mitigate the effects of failure. Some work considers replica location when scheduling tasks. Santos-neto et al. [44] schedule data-intensive jobs, and introduce *Storage Affinity*, a heuristic scheduling algorithm that exploits a data reuse pattern to consider data transfer costs and ultimately reduce job makespan.

Task replication makes copies of jobs, again for both fault tolerance and performance. Li et al. [25] strive to increase throughput and decrease Grid job 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. [26] 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. [45] 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 [28] and Sufferage [9] through simulation. Angalano et al. [5] 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. Fujimoto et al. [17] develop RR, a dynamic scheduling algorithm for independent coarse grained tasks; RR defines a ring of tasks that is scanned in round robin order to place new tasks and replicas. The authors compare their technique with five others, concluding that RR performed next to the best without knowledge of resource speed or load, even when compared with techniques that utilize such information.

Others investigate the relationship between checkpoint-restart and replication. Weissman [50] develops performance models for quantitatively comparing the separate use of the two techniques in a Grid environment. Similarly, Ramakrishnan et al. [36] compare checkpoint-restart and task replication by first analytically determining the costs of each strategy and then provide a framework that enables plug and play of resource behavior to study the effects of each fault tolerant technique under various parameters.

3 Availability Model

Resources in non-dedicated Grids oscillate between being available and unavailable to the Grid. When and how they do so depends on the failure characteristics of the machines, the policies of resource owners, the scheduling policies and mechanism of the Grid middleware, and the characteristics of the Grid's offered job load. This section identifies five availability states, and several job characteristics that could influence their ability to tolerate resource faults. We focus our discussion and analysis on Condor [27], but the results translate to any system with Condor's basic properties of (i) non-dedicated distributed resource sharing, and (ii) a mechanism that allows resource owners to dictate when and how their machines are used by the Grid.

Condor [27] harnesses idle resources from clusters, organizations, and even multi-institutional Grid environments (via flocking and compatibility with Globus [16]) by integrating resource management, monitoring, scheduling, and job queuing components. Condor can automatically create process checkpoints for migration. Condor manages non-dedicated resources, and allows individual owners to set their own policies for how and when they are utilized, as described below. Default policies dictate the behavior of resources in the absence of customized user policies, and attempt to minimize Condor's disturbance of local users and processes.

By default, Condor starts jobs only on resources that have been idle for 15 min, that are not running another Condor job, and whose local load is less than 30%. Running jobs remain subject to Condor's policies. If the keyboard is touched or if CPU load from local processes exceeds 50% for 2 min, Condor halts the process but leaves it in memory, suspended (if its image size is less than 10 MB). Condor resumes suspended jobs after 5 min of idle time, and when local CPU load falls below 30%. If a job is suspended for longer than 10 min or if its image exceeds 10 MB, Condor gives it 10 min to gracefully vacate, and then terminates it. Condor may also evict a job for a higher priority job, or if Condor itself is shut down.

Condor defaults dictate that jobs whose checkpoints exceed 60 MB checkpoint every 6 h; those with larger images checkpoint every 12 h. By default, Condor delivers checkpoints back to the machine that submits the job.

3.1 Unavailability Types

Condor's mechanism suggests a model that encompasses the following five availability states, depicted in Fig. 1:

- *Available*: An Available machine is currently running with network connectivity, more than 15 min of idle time, and a local CPU load of less than the CPU threshold. It may or may not be running a Condor job.
- *User Present*: A resource transitions to this state if the keyboard or mouse is touched, indicating that the machine has a local user.
- *CPU Threshold Exceeded*: A machine enters this state if the local CPU load increases above some owner-defined threshold, due to new or currently running jobs.

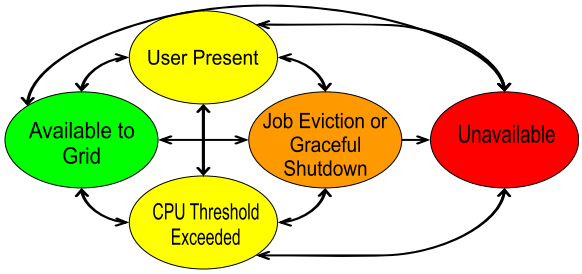

Fig. 1 Multi-state availability model: each resource resides in and transitions between five availability state, depending on the local use, reliability, and owner-controlled sharing policy of the machine

- *Job Eviction or Graceful Shutdown*: If the resource remains in the User Present or CPU Threshold Exceeded states for too long (for example for more than 15 min), if the job is evicted for any other reason, or if the machine is shut down, the resource transitions to the Job Eviction state.
- *Unavailable*: Finally, if a machine fails or becomes unreachable, it directly transitions to Unavailable.

These states differentiate the types of unavailability. If a job is running or suspended, and enters the Job Eviction state, we call this a *graceful* transition to the Unavailable state; a transition directly to Unavailable is *ungraceful*. This model is motivated by Condor's mechanism, but can reflect the policies that resource owners apply. For example, if an owner allows Condor jobs even when the user is present, the machine never enters the User Present state. Increasing local CPU threshold decreases the time spent in the CPU Threshold Exceeded state, assuming similar usage patterns. The model can also reflect the resource's job rank and suspension policy by showing when jobs are evicted directly without first being suspended.

3.2 Grid Application Diversity

Grid jobs vary in their ability to tolerate faults. A checkpointable job need not be restarted from the beginning if its host resource transitions gracefully to Unavailable. Another important factor is job runtime; Grid jobs may complete in a few seconds, or require many hours or even days [6]. Longer jobs will experience more faults, increasing the importance of their varied ability to deal with them.

Grid resources will have different characteristics in terms of how long they stay in each availability state, how often they transition between the states, and which states they transition to. Different jobs will behave differently on different resources. If a standard universe job is suspended and then eventually gracefully evicted, it could checkpoint and resume on another machine. An ungraceful transition requires using the most recent periodic checkpoint. A job that is not checkpointable must restart from the beginning, even when gracefully evicted.

The point is that job characteristics, including checkpointability and expected runtime, can influence the effectiveness of scheduling those jobs on resources that behave differently according to their transitions between the availability states identified in Section 3.1.

4 Trace Analysis

We accessed, organized, and analyzed data in the first 4 months of our 6 month Condor resource pool trace at the University of Notre Dame in early 2007. The trace consists of time-stamped CPU load (as a percentage) and idle time (in seconds). Condor records these measurements approximately every 16 min, and makes them available via the condor status command. Idle times of zero imply user presence, and the absence of data indicates that the machine was down.

The data recorded by Condor precludes determining whether the Job Eviction or Graceful Shutdown state was entered, because intentional shutdown and machine failure appear the same in the data. Since unavailability is relatively uncommon and irregular, we conservatively assume that all transitions to unavailability are ungraceful. Also, we only consider a machine to be in the user state after the user has been present for 5 min; this policy filters out short unavailability intervals that lead to job suspension, not graceful eviction. We consider a machine to be in the CPU Threshold Exceeded state if its local (i.e. non-Condor) load is above 50%. Otherwise, a machine that is online

with no user present and CPU load below 50% is considered Available. This includes machines currently running Condor jobs, which are clearly available for use by the Grid. On SMP machines, we follow Condor's approach of treating each processor as a separate resource. Our goal is to identify both the volatility of the underlying resources and any patterns of behavior in terms of availability states.

Here we examine the pool of machines as a whole, to enable conclusions about their aggregate behavior. Figure 2 depicts the number of machines in each availability state over time. Gaps in the data indicate brief intervals between the 4 months' data. The data shows a diurnal pattern; availability peaks at night, and recesses during the day, but rarely below 300 available machines. This indicates workday usage patterns and a policy of leaving machines turned on overnight, exhibited by the diurnal User Present pattern. The number of machines that occupy the CPU Threshold Exceeded state is less reflective of daily patterns. Local load appears to be less predictable than user presence, and the number of unavailable machines also does not exhibit obvious patterns. Rood et al. [41] includes further analysis of the trace.

5 Multi-State Availability Prediction

A multi-state prediction algorithm takes as input a resource and a length of time (e.g. the estimated job execution time), and produces a vector of probabilities. The vector contains one entry per availability state, and the value in each entry corresponds to the predicted probability that the resource will next enter the corresponding state. One of the entries in the vector corresponds to the Available state, and therefore contains the predicted probability that the resource will remain available throughout the interval. The probabilities of the entries in one vector sum to 100%. Our particular availability predictor outputs four probabilities, one each for entering Fig. 1's User Present, CPU Threshold Exceeded and Unavailable states next, and one for the probability of completing the interval without leaving the Available state.

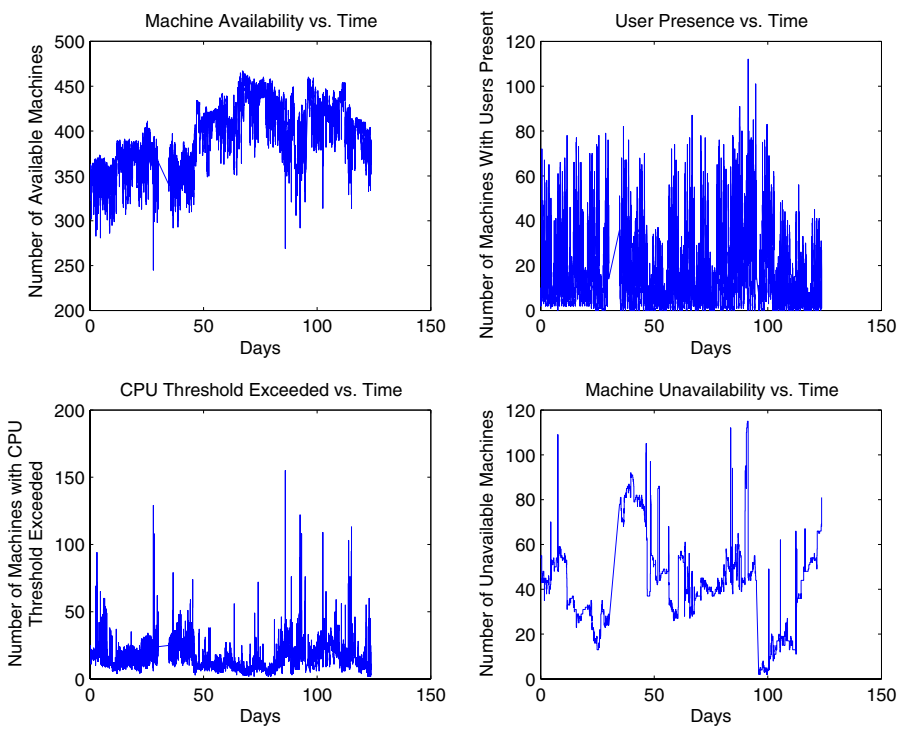

Fig. 2 Machine states over time: the four plots show the number of machines that reside in each of four different availability states (one per plot), across a 4 month trace

Since a prediction algorithm examines a resource's historical availability data, the two important aspects that differentiate predictors are (i) the *analysis interval*, which determines when during the resource's history the predictor analyzes and (ii) the *analysis technique*, which determines how the predictor analyzes that interval of availability history. We can classify our approaches along these two dimensions by combining when and how we analyze.

5.1 Analysis Interval

Previous work [38] advocates examining a resource's behavior during the interval of time that the job would run, on previous days. A different approach examines a resource's most recent activity immediately preceding the prediction, on the same day. More specifically, our predictors define the analysis intervals in two different ways:

- *N-Day* (or *Day*): This predictor examines a resource's most recent N days of availability behavior, during the interval being predicted. For example, if a job requests a 4 h job at 11am on Friday, a 3-Day predictor will base the prediction on the behavior that the resource exhibited between 11am and 2pm, on Tuesday, Wednesday, and Thursday of the same week.
- *N-Recent* (or *Recent*): This predictor examines a resource's most recent N hours of availability behavior. Thus, a 3-h predictor that schedules the same job from the example above, would consider that resource's behavior from 8am to 11am on Friday.

5.2 Analysis Techniques

Our predictors analyze the intervals using one of two different analysis techniques:

- A *Transitional* predictor counts the number of transitions from the available state, to each of the other states. For each interval that a resource was available, we count the number of times the requested job duration fits within that interval, and count that as a "transition" within the available state. We calculate the probabilities for exiting into each unavailability state, and for completing the job successfully, by summing each state's transition count and dividing each by the total number of transitions. This predictor may define a non-continuous analysis interval.
- A *Durational* predictor calculates the percentage of time that the resource resides in each of the states, and uses that percentage as the probability that it will transition to that state next. Predictors of this type will always define continuous analysis intervals leading up to the time of the prediction.

5.3 Weighting Scheme

Within the Transitional approach, we investigate two different *weighting schemes* for analyzing the analysis interval.

- *Equal Weighting* considers all transitions to have the same influence on a resource's future behavior, no matter when they occur within the analysis interval.
- *Time Weighting* considers the more recent transitions more heavily; the closer a transition is to the time of day at which the prediction occurs, the more influence that transition will have on the prediction.
- *Freshness Weighting* favors transitions that occur most recently. This differs from Time weighting when more than one day is analyzed. For example, if a prediction is made at 4pm, the Time scheme will increase the weight of transitions that were made close to 4pm on some number of previous days; the Freshness scheme considers only how long ago the transition was, not the time of day at which it occurred.

5.4 Evaluation

We evaluate Section 2's related work predictors for comparison. Since linear regression [12] is such a prevalent method in many disciplines of prediction, we have included both Sliding Window Single and Sliding Window Multi. Because states are categorical, they must be converted to numerical values. We utilize the following conventions. In the single state version of sliding window (SW

Single), we consider the Available state as 1 and any other state as 0. In the multi-state version (SW Multi), we consider the Available state as 1, User Present as 2, CPU Threshold Exceeded as 3 and Unavailable as 5. We also evaluate counter based predictors such as those used for resource availability prediction [30], including Saturating Counter and History Counter. Just as in [30], we use a 2 bit saturating counter updated every hour. The Completion predictor, which always predicts that the machine will complete the interval in the available state is used as a baseline comparison. Finally, we implement Ren's N-Day multi-state availability predictor (Ren) [39].

We evaluate our predictors for their accuracy and focus our investigation on predictors that analyze a resource's history with the Transitional analysis technique. We analyze both the N-Day and the N-Recent approaches to defining the analysis interval. For our data set, we use the 6 months of availability data from the Notre Dame trace. Each predictor performs an identical set of 500,000 predictions on a random resources at a random time during the 6 month trace. We define predictor accuracy as the ratio of correct predictions to the total number of predictions. A correct prediction is one for which the machine is predicted to exit on a certain non-available state and it does, or for which the machine is predicted to remain available throughout the interval, and it does.

5.4.1 Prediction Transition Weighting

In this section, we examine the effectiveness of three transition weighting schemes and their effect on prediction accuracy. We use a three letter naming scheme to identify our predictors.

- The first letter indicates the analysis technique; T stands for Transitional, and D would stand for Durational (we include result only for the better-performing Transitional techniques).
- The second letter indicates the method for defining the analysis interval, as described above; R stands for N-Recent, D stands for N-Day.
- The third letter identifies the weighting scheme; F stands for Freshness, E stands for Equal, and T stands for Time.

Figure 3 examines the Transitional Recent hours (TR) predictor configured to analyze the most recent N hours of availability with all three weighting schemes (Equal, Freshness and Time of day). The figure plots predictions between 5 min and 25 h (other lengths are not examined due to space constraints).

For all weighting schemes, as the predictor considers more hours, accuracy increases dramatically at first, then the increases slow down, reach a maximum, then slowly decrease. The Freshness (TRF) weighting scheme provides the best performance, reaching the highest accuracy of 77.3% when examining the past 48 h of behavior. TRF weights each transition t ($W(t)$) according to the following formula:

$$W(t) = M \cdot (T_t / L)$$

- M is the "recentness weight" set by the user (our default is five, which was derived through empirical results analysis as producing the best accuracy for this data set; other data sets may require a different value).

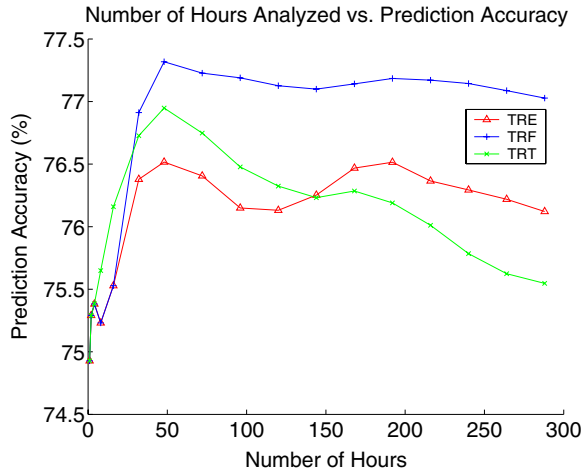

Fig. 3 Weighting scheme comparison: prediction accuracy for three different predictors, each using a different strategy for weighing the influence of transitions within a "recent-hours" analysis interval, for various numbers of hours across the X axis. TRE, TRF, and TRT use Equal, Freshness, and Time weighting, respectively

– T_t is the length of time that elapsed from the beginning of the analysis interval to the transition t.
– L is the total analysis interval length.

In the next section, we focus our analysis on the TDE and TRF predictors because they have the highest accuracy (78.3% and 77.3% respectively).

5.4.2 Prediction Accuracy Analysis

This section compares the accuracy of our best performing predictors, TDE and TRF, to several existing approaches from the literature, described in Section 2.1. In particular, we compare our work with the Saturating and History Counter predictors [29–31], the Multi-State and Single State Sliding Window predictors [12], the Ren predictor [38–40], and the Completion predictor (which simply always predicts that a job will complete in the requested interval).

Figure 4 depicts predictor accuracy versus prediction length, for predictions up to 120 h. The Counter-based predictors, Completion predictor, and Sliding Window predictors perform similarly to one another, compete well with TRF, TDE and Ren for predictions up to 20 h, and then sharply decline, never leveling off. In contrast, the TRF, TDE, and Ren predictors decrease initially and then level off; the rate of decrease for Ren is somewhat larger as the prediction length increases past 60 h. The Ren predictor initially has lower accuracy, with accuracy decreasing even faster than the Completion predictor for prediction durations shorter than 19 h. During these short intervals, the Ren predictor is up to 5.6% less accurate than the TDE predictor, and 5.2% less accurate than TRF.

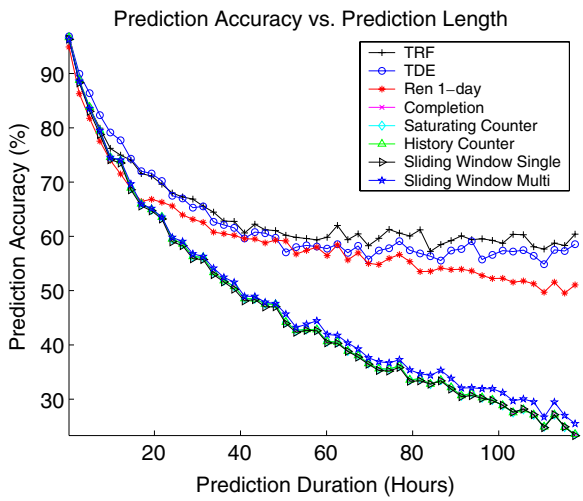

Fig. 4 Prediction accuracy by prediction duration: the accuracy of a variety of predictors for predicted job durations up to 120 h

Figure 4 demonstrates that for predictions shorter than 19 h, TDE is the most accurate, accounting for its 1% increase in accuracy over TRF for predictions between 5 min and 25 h. However, TRF becomes the most accurate predictor for predictions longer than 42 h, reaching an accuracy increase of 9.8% over Ren and 3.1% over TDE for predictions of 120 h. This large difference in accuracy for longer predictions is critical and is demonstrated in the following section. We focus our prediction-based scheduling analysis on TRF because of the improved schedules it produces. We have found that predictors that perform better for long term predictions lead to the best scheduling results when used with our scheduler, even when scheduling shorter jobs. As the results demonstrate in the following section, this explains why TRF schedules better than both TDE and Ren. For this reason, we focus on scheduling via the TRF predictor and exclude TDE from further scheduling analysis.

6 Prediction-Based Scheduling

This section investigates scheduling jobs with the aid of resource availability predictions. Section 6.1 demonstrates the inherent tradeoff between scheduling for reliability and scheduling for performance, and Section 6.2 investigates the relationship between predictor accuracy and scheduling quality.

We simulate jobs executing on machines in the Notre Dame trace, each utilizing its own recorded availability and load measurements. Our simulations create and insert each job at a random time during the 6 month simulation, such that job injection is uniformly distributed across the 6 months.

The simulation assigns each job a duration (i.e. a number of operations needed for completion) where the duration is an estimate based on the execution of the job on an unloaded resource with an average MIPS speed, where the average is calculated across all machines in the trace. Application durations are uniformly distributed across different intervals, depending on the test. This uniform distribution of job insertion times and durations allows us to test the quality of the predictor and scheduler combination with equal weight for all prediction durations and prediction start times, without emphasizing a particular duration or start time.

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 jobs. Resources are considered available if they are running, connected, have no user present and have a local CPU load below 30%. A resource may only be assigned one task for execution at a time. During each simulator tick, the simulator calculates the number of operations completed by each working resource, and updates the records for each executing job. If a resource executing a job leaves the Available state (as per the trace), effectively evicting the job, the executing job joins the back of the job queue for rescheduling. The number of operations that remain to be completed for this evicted job depends on whether the job is checkpointable, and on the type of eviction, as described in Section 3. Non-checkpointable jobs that are evicted must restart from the beginning of their execution, regardless of the type of eviction. All evicted jobs are added to the job queue and immediately rescheduled on an available resource. Once every 3 min, the simulator repeatedly removes and schedules jobs from the head of the queue until no more jobs can be scheduled.

To facilitate prediction-based scheduling, we define a scheduling algorithm called *Prediction Product Score (PPS) Scheduler*. The PPS Scheduler scores each available resource and maps the job at the head of the queue onto the resource with the highest score (Ties are broken arbitrarily.)

We analyze scheduling quality according to average job makespan and evictions. Makespan is the time from submission to completion; average makespan is calculated across all jobs in the simulation. For evictions, the simulator counts the total number of times that jobs need to be rescheduled because a resource running a job transitions from available to one of the unavailable states.

6.1 Reliability Performance Relationship

This section establishes the inherent tradeoff between scheduling for reliability and for performance. Schedulers that favor performance consider the static capability of target machines, along with current and predicted load conditions, to decrease makespan. Other schedulers may instead consider the past reliability and behavior of machines to predict future availability and to increase the number of jobs that complete successfully without interruption due to machines failing or owners reclaiming resources. Schedulers may consider both factors, but cannot in general optimize simultaneously both for performance-based metrics like makespan and for reliability-based metrics like the number of evictions or the number of operations that must be re-executed due to eviction (i.e. "operations lost") [13].

To investigate the tradeoff, we ust the PPS scheduler and score resources according to the following expression:

$$RS_i = (1 - W) \cdot P_i[COMPLETE] + W \cdot (MIPSi/MIPS_max) \cdot (1 - L_i)$$

- $P_i[COMPLETE]$ is resource i's predicted probability of completing the job interval without failure, according to the TRF predictor,
- $MIPS_i$ is the resource's processor speed,
- $MIPS_max$ is the highest processor speed of all resources (for normalization), and
- L_i is the resource's current processor load.

Reliability influences completion probability $P_i[COMPLETE]$, performance influences $(MIPS_i/MIPS_max) \cdot (1 - L_i)$, and the Tradeoff Weight (W) determines which more heavily influences the resource's overall score.

We simulate executing 6,000 jobs with durations uniformly distributed between 5 min and 25 h, 25% of which are checkpointable (the remaining 75% are non-checkpointable).

Figure 5 illustrates the effect of varying the Tradeoff Weight and hence the relative influence of reliability or performance in scheduling. As performance is considered more prominently, makespan decreases and the number of evictions increases. In the middle of the plot, a tradeoff weight of 0.5 does achieve makespan within 6.7% of the lowest makespan on the curve, while simultaneously coming within 18.1% of the fewest number of evictions. Nevertheless, the makespan slope is uniformly negative, and the evictions slope is uniformly positive.

6.2 Prediction Quality Versus Scheduling Results

This section investigates the effect of predictor accuracy on scheduler performance. We modify the PPS Scheduler's resource scoring expression as follows:

$$RS_i = MIPS_i \cdot (1 - L_i) \cdot P_i[COMPLETE]$$

- $P_i[COMPLETE]$ is resource i's predicted probability of completing the job interval without failure, according to the TRF predictor,
- $MIPS_i$ is the resource's processor speed, and
- L_i is the resource's current processor load.

This scoring expression incorporates both reliability and performance but does not include a variable tradeoff weight.

We also modified the PPS scheduler to vary the prediction duration. Previously, the scheduler had asked the predictor about behavior over a prediction interval that is intended to match job runtime. However, a prediction for the next N hours may not necessarily reflect the best information for scheduling an N hour job. We have previously investigated the effect of scheduling an N hour job using predictions made for the next $(M \cdot N)$ hours, where M is the "interval multiplier." Our research developed two "hybrid" multipliers:

- The *Comparison Scheduler* (*Comp* or *TRF-Comp*) uses M = 3 for jobs less than 28 h, and M = 0.25 for longer jobs.
- The *Performance Scheduler* (*Perf* or *TRF-Perf*) ignores reliability on jobs less than 28 h, instead selecting the fastest, least loaded resources; for longer jobs, it uses M = 0.25.

We compare the two multi-state based schedulers, namely the TRF predictor coupled with the PPS scheduler, and the Ren MTTF prediction-based scheduler, with one another (Section 6.2.1)

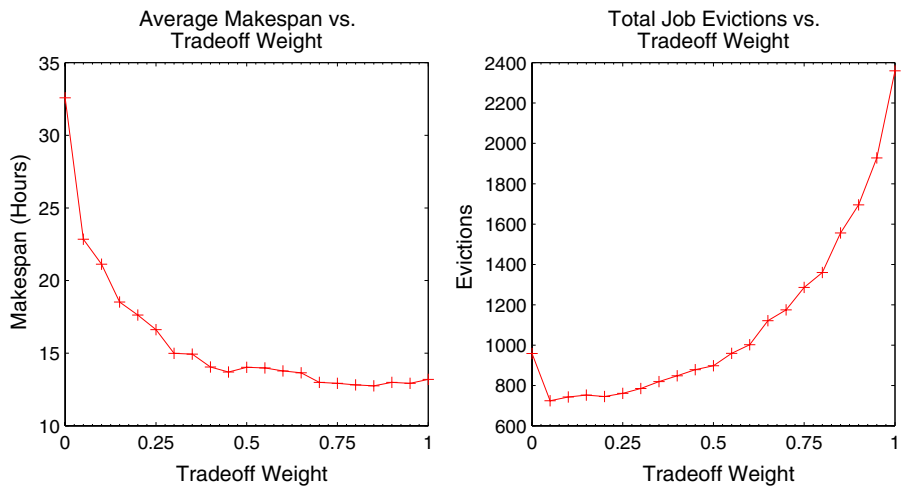

Fig. 5 Reliability performance tradeoff: makespan and evictions for various tradeoff weights in the PPS Scheduler's resource ranking formula

and then against other scheduling techniques (Section 6.2.2).

6.2.1 Multi-State Prediction Based Scheduling

This section explores our Transitional Recent-hours Freshness-weighted (TRF) scheduler and Ren's MTTF scheduler.

We vary the number of days that Ren's predictor uses to make its predictions, while simultaneously varying TRF's interval multiplier for comparison. These parameters allow each scheduler to trade off reliability and performance. Again, we simulate 6,000 jobs inserted at random times during the 6 month trace, with durations uniformly distributed between 5 min and 25 h; 25% of the jobs are checkpointable.

Figure 6 shows the average makespan and number of evictions obtained and caused by TRF and Ren MTTF as we vary the interval multiplier for TRF and the number of days analyzed for Ren MTTF. Varying each parameter allows that scheduler to trade off performance (lower makespan) for reliability (fewer evictions). In selecting a point for Ren MTTF on one curve, however, TRF does better in terms of the other metric. For example, for approximately 1,500 evictions, TRF has average makespan that is 27% lower, and for a makespan of 13 h, TRF has 52% fewer evictions.

6.2.2 Scheduling Quality Analysis

This section further compares our best performing scheduling approaches, TRF-Comp and TRF-Perf, to other scheduling methods. To more thoroughly understand the characteristics of these schedulers in a variety of conditions, we perform tests with a diverse set of job lengths. We report results for simulating 6,000 jobs, 25% of which are checkpointable, over the 6 month Notre Dame trace. The jobs range from 5 min to the job length indicated on the x-axis.

Figure 7's top two graphs compare the TRF schedulers to Ren-MTTF (with four different

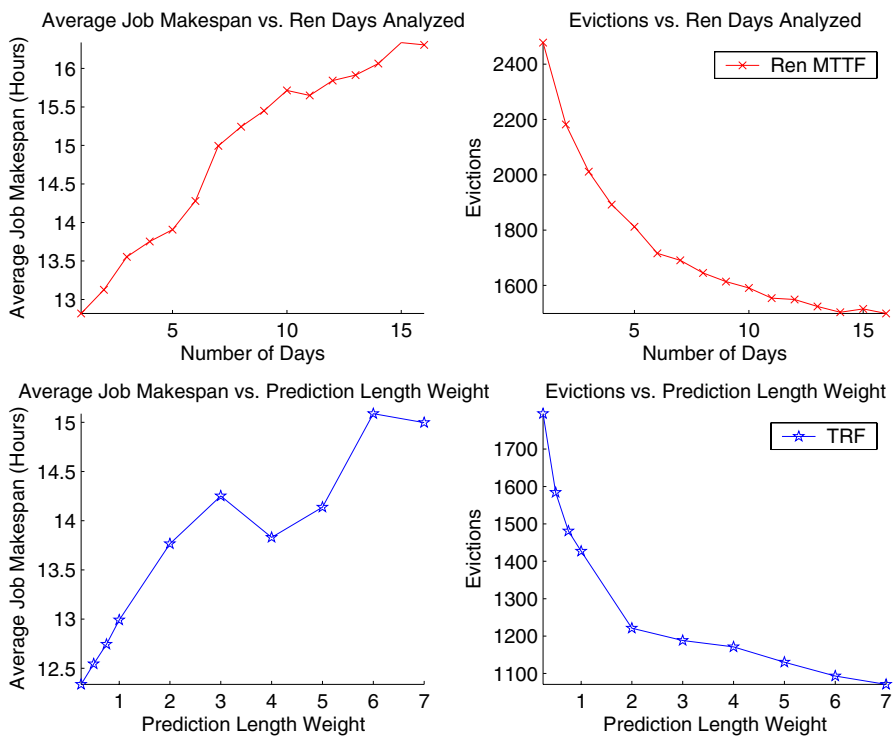

Fig. 6 TRF scheduler vs. Ren MTTF scheduler, in terms of average makespan (*plots on the left*) and overhead in terms of evictions (*plots on the right*). The top two plots are for Ren MTTF, the bottom two for TRF. The X axis is different for the two pairs of graphs; each shows results across a range of values for the parameter that allows that scheduler to trade off between reliability and performance

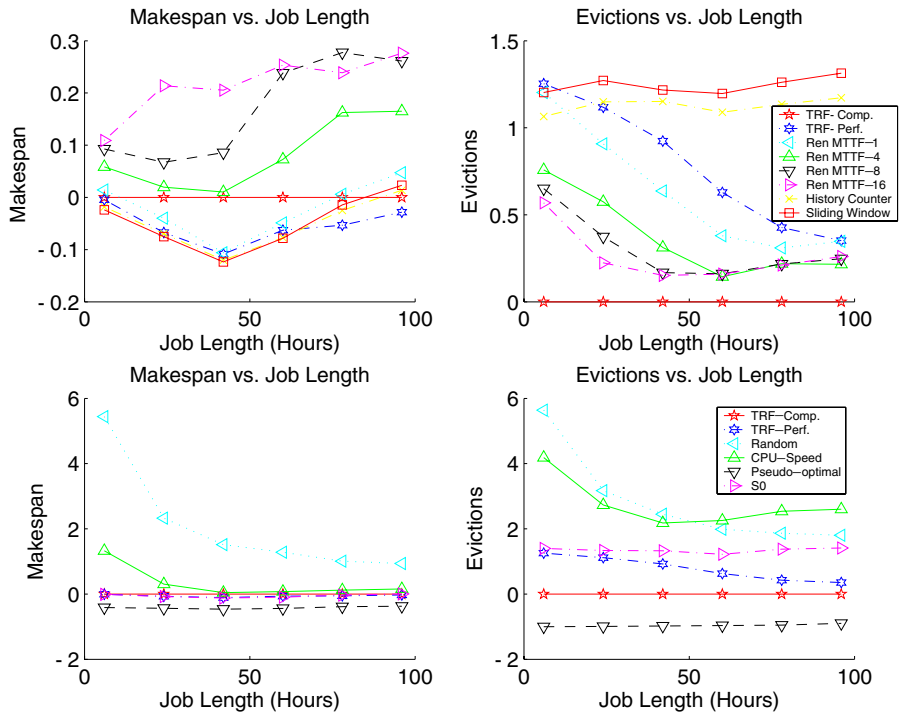

Fig. 7 Scheduling results across job lengths: makespan and evictions for a variety of Schedulers across a range of job lengths

numbers of days analyzed) and to the History and Sliding Window prediction-based schedulers. The graphs plot the percentage difference in both makespan and the number of evictions, compared with TRF-Comp. The History and Sliding Window predictors utilize the Comparison Scheduler (Comp) as well.

TRF-Comp maintains comparable makespan as job length increases, peaking at roughly 11% higher than Sliding Window, History, TRF-Perf and Ren MTTF-1 for jobs of up to 40 h in length. For this same length, TRF-Comp achieves 60% fewer evictions than the next most reliable scheduler, Ren MTTF-1. For all job lengths up to 40 h, TRF-Comp achieves at least 15% fewer evictions when compared with the most reliable scheduler, Ren MTTF-16; average job makespan simultaneously decreases by 20% (27.3 h versus 32.9 h). TRF-Comp also decreases the number of evictions by at least 57% compared with all other schedulers, for jobs up to 6 h long (355 evictions versus 557). TRF-Perf comes within 1% of the shortest makespan (Sliding Window) for shorter lengths, and achieves the shortest makespan for jobs of 80+ h.

Figure 7's bottom two graphs compare TRF-Comp and TRF-Perf with the following non-prediction-based scheduling approaches:

- *Random* selects an available resource at random,
- *CPU Speed* selects the resource with the fastest CPU speed,
- *Pseudo Optimal* selects the available resource that will execute the job in the smallest execution time, without failure, based on omnipotent future knowledge of resource availability. When all machines would fail before completing the job, the Pseudo Optimal Scheduler chooses the fastest available resource in terms of MIPS speed, and
- *S0* considers the speed and load of the resource, by multiplying the MIPS score with one minus the current resource load [42]: $MIPS_i \cdot (1 - L_i)$

For average job makespan, TRF-Comp follows the non-optimal schedulers and produces the fewest evictions for all job lengths, by at least

110%. Elsewhere we report that TRF-Comp also produces the best results for a variety of loads [42].

7 Replication

Checkpointing and replication are two of the primary tools for dealing with resource unavailability. The system cannot generally add checkpointability; that's up to the application programmer. So whereas schedulers can (and should) take advantage of knowing which jobs are checkpointable, they cannot pro-actively 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 job makespan is determined by the earliest completion time among all replicas; this can depend on unpredictable load and usage patterns. Secondly, replicated job executions can also help deal with failure; then when one resource fails, the adverse effect on the performance of the jobs it runs can be reduced if others have a chance to complete those jobs.

Replication does not come without a cost, however. Within a Grid economy, it is likely that jobs 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 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.

This section investigates the effect that replicating jobs can have on average job makespan and the number of extra operations performed (overhead). We explore three classes of replication techniques:

- *Static* techniques (Section 7.1) replicate based on the characteristics of the job being scheduled,
- *Prediction-Based* techniques (Section 7.2) consider forecasts about the future availability and load of resources, and
- *Load Adaptive* techniques (Section 7.4) change their behavior based on observed system load.

All experiments use the PPS Scheduler described in Section 6, augmented to support replication. Upon placing each job, the scheduler uses the replication policy to determine how many replicas to make. In this paper, the scheduler makes 0 or 1 replicas. When a task or its replica completes, the system terminates other copies, freeing up the resources executing them. Schedulers whose replication policies require an availability prediction use the TRF predictor, unless we specify otherwise.

We analyze scheduling quality according to average job makespan, extra operations, and replication efficiency, which is defined and discussed later. Task lengths are defined in terms of the number of operations needed to complete them; *extra operations* refers to the number of additional operations that the system performs for a job, including all "lost" operations due to eviction, and any operations performed by replicas (or initially scheduled jobs) that do not ultimately finish because some other copy of that job finished first.

7.1 Static Replication Techniques

We first explore the effect that replicating jobs based on checkpointability can have on both extra operations and on job makespan. We vary the total number of jobs, using 1K, 14K, 26K, and 40K total jobs over the 6 month trace, in four separate sets of simulations. This translates to 0.001, 0.13, 0.26, and 0.39 jobs per resource per day, respectively. We refer to these tests as the *Low*, *Medium Low*, *Medium High*, and *High* load cases. For this test, we execute 6,000 jobs, half of which are checkpointable.

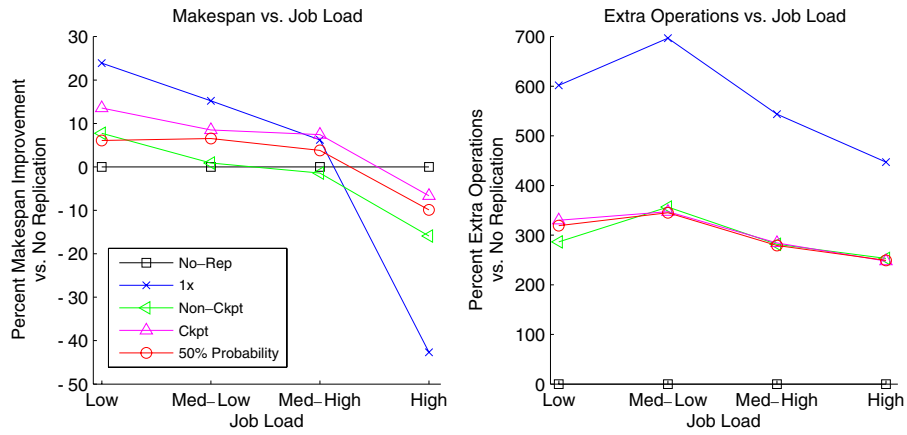

Fig. 8 Checkpointability based replication: makespan (*left*) and extra operations (*right*) for a variety of replication strategies, two of which are based on checkpointability of jobs, across four different load levels

Figure 8 includes the following replication policies:

- *1x*: Replicates each job exactly once. If either the main job or the replica runs on a resource that becomes unavailable, it is rescheduled but never re-replicated. Thus, two versions of each job are always running.
- *Non-Ckpt*: Replicates only jobs that are not checkpointable.
- *Ckpt*: Replicates only jobs that are checkpointable.[1]
- *50% Probability*: Replicates half of the jobs, at random.
- *No-Rep*: Does not make replications

Figure 8 illustrates that under low loads, increasing replicas achieves more makespan improvement, a benefit that falls off for higher loads because replication forces subsequent jobs to use even less desirable resources. Replicating only non-checkpointable jobs (Non-Ckpt) improves makespan for all but the highest load test, and at significantly less cost than 1x replication. Moreover, replicating the half of the jobs that are non-checkpointable is better than replicating half of the jobs at random, which indicates that using checkpointability for replication policies has benefit, as expected.

[1]This replication policy is not based on intuition, but instead serves as a useful comparison for the Non-Ckpt policy

7.2 Prediction-Based Replication Techniques

Ideally, schedulers would replicate only those jobs that are most likely to fail. In deciding whether to replicate jobs based on predicted resource reliability, we first schedule the jobs on the "best available" resource. For this work, we select a resource based on its projected near-future performance (speed and load) only, and replicate based on the probability of completing the job on that resource. The *Resource Score* is calculated as follows:

$$RS_i = MIPS_i \cdot (1 - L_i)$$

$MIPS_i$ is the resource's score and L_i is its current processor load. We ran a mirrored set of tests to consider speed and reliability in initial scheduling, and confirmed that when task replication is used for reliability, better results stem from performance-based scheduling.

We propose a method for considering makespan improvement and overhead within the same metric, to quantify performance improvement per replica. We define *Replication Efficiency (RE)* as:

$$\text{RE} = \text{Makespan Improvement}/\text{Replicas per Job}$$

Makespan Improvement is the improvement over not creating any replicas

Increasing makespan improvement with the same number of replicas will increase efficiency, and increasing replicas to get the same makespan improvement will decrease efficiency. The best replication strategies will make replicas of the

"right" jobs, and achieve more improvement for the same cost (number of replicas).

Replication efficiency allows users to quantify the tradeoff between reliability and performance. Frugal users may require high replication efficiency, whereas performance minded users may care only about makespan improvement. Replication Efficiency can also be useful to administrators in choosing which replication policy is most suited to the goals of their system.

For scheduling, each scheduler maintains a *Replication Score Threshold (RST)* that it compares against a resource's predicted probability of completion, at the time it considers placing a job on that resource. An RST-based scheduler will make a replica of the job if the completion probability falls below the threshold.

We investigate the following replication strategies:

- *RST Scheduler*: Replicates all jobs that are scheduled on resources whose predicted probability of completion without interruption falls below the Replication Score Threshold (RST) with which the scheduler is configured.
- *RST & NC & Len*: Replicates all non-checkpointable jobs that are longer than 10 h, and whose predicted completion probability falls below the RST.
- *1x*: Makes exactly one replica of each job.
- *NC & Len*: Replicates non-checkpointable jobs longer than 10 h

We vary the RST value to study its effect on makespan improvement and replication efficiency. The RST Scheduler configured with an RST of 100 replicates all jobs scheduled on resources with a predicted completion probability below 100%. These predictions are made by the TRF Predictor on the selected resource.

Figure 9 plots Makespan and Replication Efficiency versus Replication Score Threshold (RST) in the low load case. The left graph of Fig. 9 shows makespan improvement over not replicating, across a range of RST values. Schedulers with larger RST values make more replicas. Under low loads, this aggressive replication strategy improves makespan. The 1x strategy achieves the highest makespan improvement, as it creates replicas of all jobs. In terms of replication efficiency, lower RST values perform better, and the 1x strategy has among the lowest replication efficiencies. RST &NC &Len's efficiency outperforms NC &Len due to its use of resource availability predictions. This indicates that the better the scheduler is at selecting the right jobs to replicate, the higher the achieved efficiency.

7.3 Replication Quality Versus Prediction Accuracy

This section investigates the effect that prediction accuracy has on the quality of the replication decisions. We compare the TRF predictor with the Ren predictor; both use the same replication

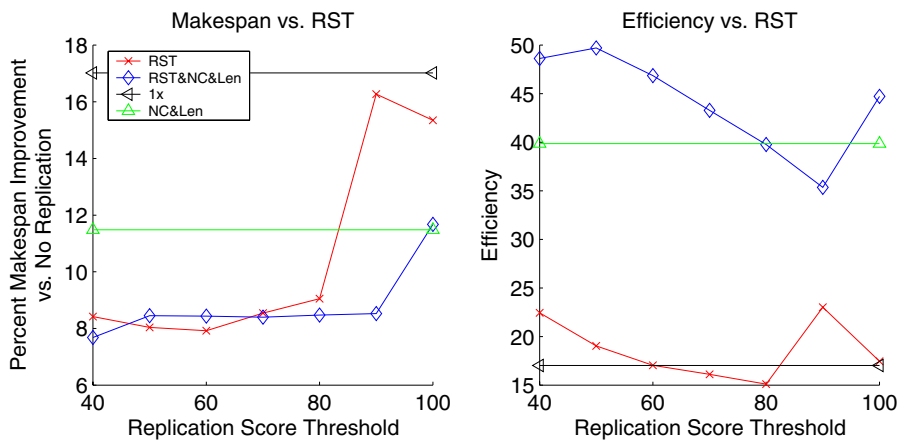

Fig. 9 Makespan and efficiency versus replication score threshold under low load, for four different replication schedulers

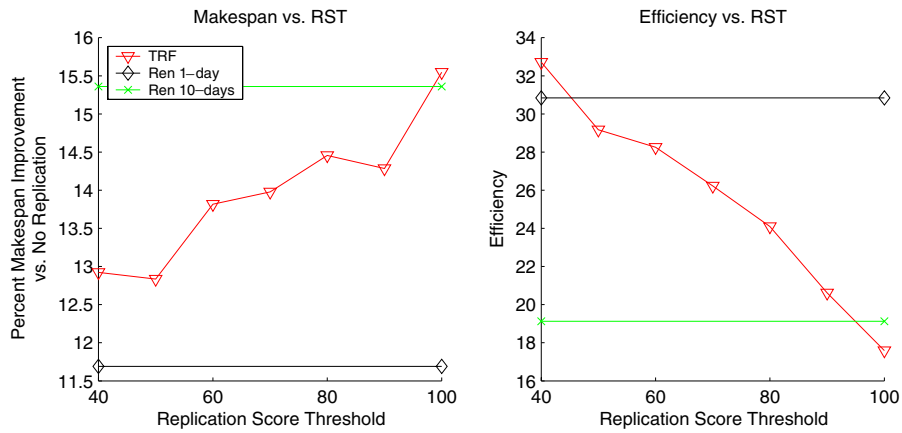

Fig. 10 TRF vs. Ren predictor, makespan and efficiency for a variety of RST values

strategy to isolate the effect of accuracy. The Ren predictor returns only whether or not it believes the job will complete in the requested interval, not the probability of doing so. Because the RST Scheduler depends on a predicted probability, we cannot use the Ren predictor with it, in the same way. As discussed previously, though, varying the number of days that the Ren predictor analyzes introduces the same tradeoff between makespan and efficiency as varying the RST of TRF.

Figure 10 plots makespan and replication efficiency versus RST. We plot TRF, Ren 1-day, and Ren 10-day. We omit other Ren day counts as they produce consistent intermediate results. TRF with an RST of 100 achieves the largest makespan increase among all predictors. TRF with an RST = 40 produces the highest efficiency. Other RST values allow the schedule to effectively navigate the tradeoff between performance and efficiency.

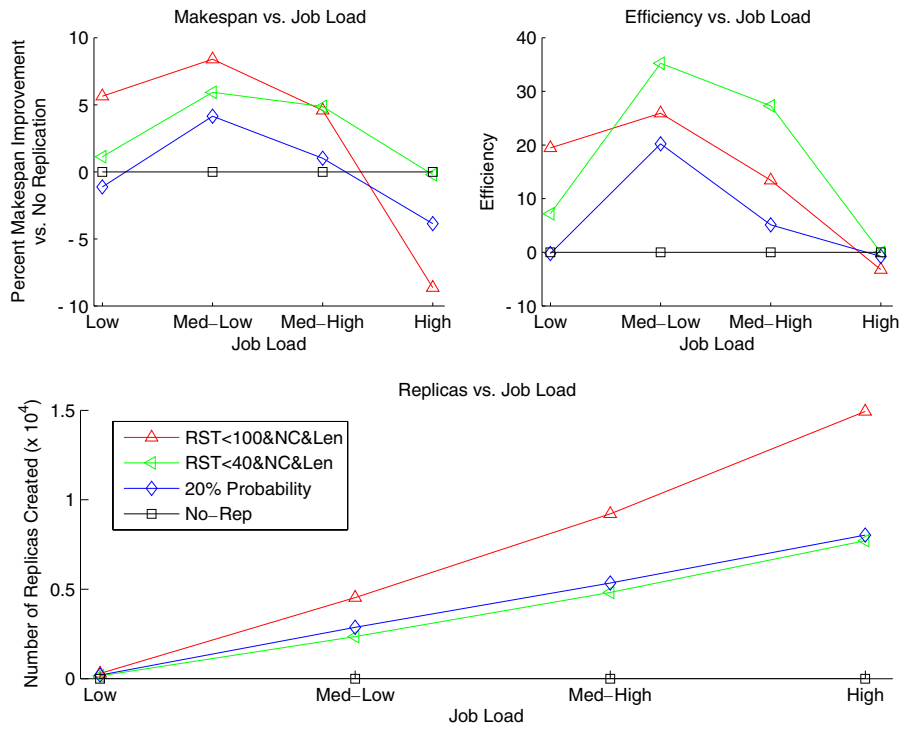

Fig. 11 Comparison of prediction-based schedulers: makespan improvement, replication efficiency, and number of replicas created for four different schedulers

7.4 Load Adaptive Replication Techniques

This section investigates the effect that varying system load has on the performance achieved by the replication techniques proposed in the previous section. Figure 11 plots our RST&NC&Len replication technique with RST values of both 100% and 40% and compares them with not replicating (No-Rep) and with replicating a random 20% of jobs (20% Probability). As load increases, the makespan improvement and efficiency of all replication strategies initially increases slightly but then falls under Med-High and High loads. Under Low loads, replicas are less likely to interfere with other resource by taking up the only available high-quality machines. As load increases, fewer machines are available; if the system uses them for replicas, other jobs must wait or run on the very slowest machines that are the most likely to fail.

In comparing our results to blind (random) replication, the RST<40&NC&Len technique creates roughly the same number of replicas as randomly replicating 20% of jobs. Given the same number of replicas, the RST technique achieves a higher makespan improvement and efficiency across all load levels. This demonstrates that it is choosing the "right" jobs to replicate when compared with randomly replicating jobs.

Figure 11 also demonstrates that the replication strategy that provides the highest achievable makespan or efficiency varies based on the load of the system. In particular, either RST<40&NC&Len or RST<100&NC&Len produces the largest makespan improvement. This suggests a replication strategy that behaves differently depending on the load.

Three load adaptive replication techniques respond to varying load levels with three different replication strategies, to achieve different desired metrics. To determine the current load level, the schedulers track the number of jobs submitted to the system in the past day. The load adaptive replication strategies are *Makespan*, *Efficiency*, and *Compromise*. They behave as follows:

- *Makespan*: Maximize 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-Low load*: RST<40 - Replicate a job if the completion probability is less than 40% and the job is non-checkpointable
  - *High load*: No replication
- *Efficiency*: Maximize replication efficiency across loads
  - *All loads*: RST<40&Len&NC - Replicate a job if the completion probability is less than 40%, the job is non-checkpointable and the length is over 10 h

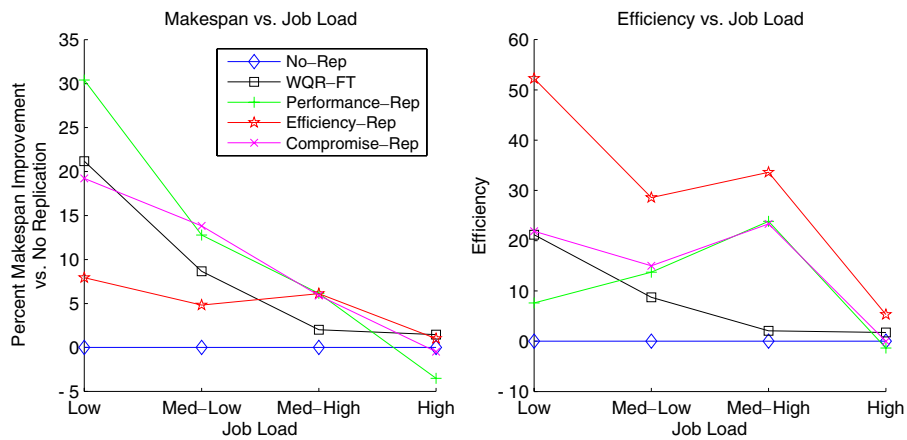

Fig. 12 Load adaptive replication: three different schedulers satisfy different performance and efficiency metrics under four different loads

- *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-Low 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

Figure 12 compares the three load adaptive policies across all four load levels. The Makespan technique produces the largest makespan improvement, but with the lowest efficiency (other than the high load case, when the other strategies make similar numbers of replicas). The unpredictability of high load cases result in small and variable makespan increases and varied replication efficiency. This situation aside, all techniques do meet their goals. The Efficiency technique produces the largest efficiency across all but the high load case, while simultaneously producing the smallest makespan improvement. The Compromise technique produces an average makespan improvement and replication efficiency across all loads.

8 Summary

Scheduling in a large scale Grid that comprises a heterogeneous collection of eclectic resources requires techniques for dealing with resource failure and unavailability. Availability predictors can forecast resource behavior, allowing schedulers to consider reliability in conjunction with performance and load, when placing jobs. This paper describes and compares several prediction techniques and shows that our predictors result in the highest achieved accuracy. The paper also describes schedulers that utilize these availability predictions to make better placement decisions for jobs. Our prediction-based scheduling strategies, when compared with traditional and related work prediction techniques, both reduce average job makespan and decrease the number of job evictions. Sometimes, however, simply finding the "best" resource, even by more sophisticated selection criteria, is not enough.

Schedulers can replicate jobs to reduce the effect of failure and to ultimately reduce makespan. Replication requires additional Grid cycles, which can have direct cost within a Grid economy, and indirect cost in tying up attractive resources, especially under relatively higher loads. This paper describes techniques that use availability predictions to influence replication decisions. A strategy that considers process checkpointability, job length, and predicted resource reliability does the best job of using additional replicated operations to reduce job makespan.

References

1. Abu-Ghazaleh, N., Lewis, M.: Toward self organizing Grids. In: International Conference on High Performance Distributed Computing Hot Topics Session, pp. 324–327 (2006)
2. Amin, A., Ammar, R., Gokhale, S.: An efficient method to schedule tandem of real-time tasks in cluster computing with possible processor failures. In: Symposium on Computers and Communications, p. 1207 (2003)
3. Anderson, D.: Boinc: a system for public-resource computing and storage. In: IEEE/ACM Workshop on Grid Computing, pp. 4–10 (2004)
4. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution tech. J. Am. Coll. Med. Coding Spec. **36**(4), 335–371 (2004)
5. Anglano, C., Canonico, M.: Fault-tolerant scheduling for bag-of-tasks Grid applications. In: Advances in Grid Computing - EGC 2005, pp. 630–639 (2005)
6. Arpaci, R., Dusseau, A., Vahdat, A., Liu, L., Anderson, T., Patterson, D.: The interaction of parallel and sequential workloads on a network of workstations. In: International Conference on Measurement and Modeling of Computer Systems, pp. 267–278 (1995)
7. Braun, T., Siegel, H., Beck, N.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
8. Cardinale, Y., Casanova, H.: An evaluation of job scheduling strategies for divisible loads on Grid platforms. In: High Performance Computing and Simulation Conference, pp. 705–712 (2006)
9. Casanova, H., Zagorodnov, D., Berman, F., Legrand, A.: Heuristics for scheduling parameter sweep applications in Grid environments. In: HCW '00: Proceed-

ings of the 9th Heterogeneous Computing Workshop, p. 349. IEEE Computer Society, Washington, DC (2000)
10. Chun, B., Vahdat, A.: Workload and failure characterization on a large-scale federated testbed. Technical Report IRB-TR-03-040, Intel Research Berkeley (2003)
11. Dail, H., Casanova, H., Berman, F.: A decoupled scheduling approach for Grid application development environments. J. Parallel Distrib. Comput. **63**(5), 505–524 (2003)
12. Dinda, P., O'Hallaron, D.: An extensive toolkit for resource prediction in distributed systems. Technical Report CMU-CS-99-138, Carnegie Mellon University (1999)
13. Dogan, A., Ozguner, F.: Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. Comput. J. **48**(3), 300–314 (2005)
14. E.G. for EsciencE: E.G. for EsciencE homepage. http://public.eu-egee.org/ (2008)
15. Foster, I., Iamnitchi, A.: On death, taxes, and the convergence of peer-to-peer and Grid computing. In: International Workshop on Peer-To-Peer Systems (2003)
16. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-g: a computation management agent for multi-institutional Grids. In: International Conference on High Performance Distributed Computing, pp. 55–63 (2001)
17. Fujimoto, N., Hagihara, K.: A comparison among Grid scheduling algorithms for independent coarse-grained tasks. In: International Symosium on Applications and the Internet, pp. 674–680. IEEE Computer Society, Washington, DC (2004)
18. O.S. Grid: O.S. Grid homepage. http://www.opensciencegrid.org/ (2008)
19. Kang, W., Grimshaw, A.S.: Failure prediction in computational Grids. In: Simulation Symposium, pp. 275–282 (2007)
20. Kartik, S., Murthy, C.: Task allocation algorithms for maximizing reliability of distributed computing systems. IEEE Trans. Comput. **41**(9), 1156–1168 (1992)
21. Kondo, D., Anderson, D., McLeod, J.: Performance evaluation of scheduling policies for volunteer computing. In: International Conference on e-Science, pp. 415–422 (2007)
22. Kondo, D., Chien, A., Casanova, H.: Resource management for rapid application turnaround on enterprise desktop Grids. In: International Conference on High Performance Computing, p. 17 (2004)
23. Lamehamedi, H., Szymanski, B., Shentu, Z.: Data replication strategies in Grid environments. In: in Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP02), pp. 378–383. Press (2002)
24. Lewis, M., Grimshaw, A.: The core legion object model. In: International Conference on High Performance Distributed Computing, pp. 551–561 (1996)
25. Li, Y., Mascagni, M.: Improving performance via computational replication on a large-scale computational Grid. In: CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid, p. 442. IEEE Computer Society, Washington, DC (2003)
26. Litke, A., Skoutas, D., Tserpes, K., Varvarigou, T.: Efficient task replication and management for adaptive fault tolerance in mobile Grid environments. Future Gener. Comput. Syst. **23**(2), 163–178 (2007)
27. Litzkow, M., Livny, M., Mutka, M.: Condor—a hunter of idle workstations. In: International Conference on Distributed Computing Systems, pp. 104–111 (1988)
28. Menascé, D.A., Saha, D., da Silva Porto, S.C., Almeida, V.A.F., Tripathi S.K.: Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. Parallel J. Distrib. Comput. **28**(1), 1–18 (1995)
29. Mickens, J., Noble, B.: Predicting node availability in peer-to-peer networks. In: International Conference on Measurement and Modeling of Computer Systems (2005)
30. Mickens, J., Noble, B.: Exploiting availability prediction in distributed systems. In: Network Systems Design and Implementation, pp. 73–86 (2006)
31. Mickens, J., Noble, B.: Improving distributed system performance using machine availability prediction. In: International Conference on Measurement and Modeling of Computer Systems Performance Evaluation Review, vol. 34(2) (2006)
32. Nurmi, D., Brevik, J., Wolski, R.: Modeling machine availability in enterprise and wide-area distributed computing environments. In: Europar, pp. 432–441 (2005)
33. Planetlab: P. L. A. open platform for developing debugging and accessing planetary scale services. http://www.planet-lab.org/ (2008)
34. Pietrobon, V., Orlando, S.: Performance fault prediction models. Technical Report CS-2004-3, University of Venice (2004)
35. Qin, X., Jiang, H., Xie, C., Han, Z.: Reliability-driven scheduling for real-time tasks with precedence constraints in heterogeneous distributed systems. In: International Conference on Parallel and Distributed Computing, pp. 617–623 (2000)
36. Ramakrishnan, L., Reed, D.A.: Performability modeling for scheduling and fault tolerance strategies for scientific workflows. In: HPDC '08: Proceedings of the 17th International Symposium on High Performance Distributed Computing, pp. 23–34. ACM, New York (2008)
37. Ranganathan, K., Foster, I.: Identifying dynamic replication strategies for a high performance data Grid. In: In Proc. of the International Grid Computing Workshop, pp. 75–86 (2001)
38. Ren, X., Eigenmann, R.: Empirical studies on the behavior of resource availability in fine-grained cycle sharing systems. In: International Conference on Parallel Processing, pp. 3–11 (2006)
39. Ren, X., Lee, S., Eigenmann, R., Bagchi, S.: Resource failure prediction in fine-grained cycle sharing system. In: International Conference on High Performance Distributed Computing (2006)
40. Ren, X., Lee, S., Eigenmann, R., Bagchi, S.: Prediction of resource availability in fine-grained cycle sharing

systems empirical evaluation. Journal of Grid Computing **5**(2), 173–195 (2007)
41. Rood, B., Lewis, M.: Multi-state Grid resource availability characterization. In: International Conference on Grid Computing, pp. 42–49 (2007)
42. Rood, B., Lewis, M.: Scheduling on the Grid via multi-state resource availability prediction. In: International Conference on Grid Computing (2008)
43. Sahoo, R., Oliner, A., Rish, I., Gupta, M., Moreira, J., Ma, S., Vilalta, R., Sivasubramaniam, A.: Critical event prediction for proactive management in large-scale computer clusters. In: Special Interest Group on Knowledge Discovery and Data Mining, pp. 426–435 (2003)
44. Santos-neto, E., Cirne, W., Brasileiro, F., Lima, R., Grande, C.: Exploiting replication and data reuse to efficiently schedule data-intensive applications on Grids. In: Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing, pp. 210–232 (2004)
45. Silva, D.P.D., Cirne, W., Brasileiro, F.V., Grande, C.: Trading cycles for information: using replication to schedule bag-of-tasks applications on computational Grids. In: Applications on Computational Grids, in Proc of Euro-Par 2003, pp. 169–180 (2003)
46. Srinivasan, S., Jha, N.: Safety and reliability-driven task allocation in distributed systems. In: International Conference on Parallel and Distributed Systems, pp. 238–251 (1999)
47. Teragrid: Teragrid homepage. http://www.teragrid.org (2008)
48. Vilalta, R., Ma, S.: Predicting rare events in temporal domains. In: International Conference on Data Mining, p. 474 (2002)
49. Weiss, G., Hirsh, H.: Learning to predict rare events in categorical time-series data. In: International Conference on Machine Learning, pp. 83–90 (1998)
50. Weissman, J.B.: Fault tolerant computing on the Grid: what are my options. Technical report, University of Texas at San Antonio (1998)
51. Wolski, R., Spring, N., Hayes, J.: The network weather service: a distributed resource performance forecasting service for metacomputing. Future Gener. Comput. Syst. **15**, 757–768 (1999)