Scheduling on the Grid via Multi-State Resource Availability Prediction

Brent Rood and Michael J. Lewis
Department of Computer Science
State University of New York at Binghamton
Binghamton, NY 13902
{brood1, mlewis}@binghamton.edu

Abstract

To make the most effective application placement decisions on volatile large-scale heterogeneous Grids, schedulers must consider factors such as resource speed, load, and reliability. Including reliability requires availability predictors, which consider different periods of resource history, and use various strategies to make predictions about resource behavior. Prediction accuracy significantly affects the quality of the schedule, as does the method by which schedulers combine various factors, including the weight given to predicted availability, speed, load, and more. This paper explores the question of how to consider predicted availability to improve scheduling, concentrating on multi-state availability predictors. We propose and study several classes of schedulers, and a method for combining factors. We characterize the inherent tradeoff between application makespan and the number of evictions due to failure, and demonstrate how our schedulers can navigate this tradeoff under various scenarios. We vary application load and length, and the percentage of jobs that are checkpointable. Our results show that the only other multi-state prediction based scheduler causes up to 51% more evicted jobs while simultaneously increasing average job makespan by 18% when compared with our scheduler.¹

1. Introduction

Making the most efficient and effective use of computational Grid resources requires Grid schedulers that choose the "right" resources for the applications that users submit. Schedulers may attempt to decrease job makespan [15], increase the overall Grid throughput [5], or even share or balance the load across sites or machines. Schedulers may also attempt to increase the reliability of Grid applications [24], or to minimize the cost of job execution within a computational Grid economy [7].

These different metrics require various supporting mechanisms, including resource load monitors [4] and predictors [25], availability histories [21], and predictors [20][22]. As Grids evolve to include volatile resources owned by individuals at one extreme, alongside dedicated and highly-available high-performance clusters at the other, Grid schedulers must become more sophisticated to achieve multiple goals in this increasingly heterogeneous environment.

In particular, schedulers must effectively predict the availability of constituent resources, and use these predictions to make scheduling decisions. Ignoring resource reliability characteristics can lead to longer application makespans due to wasted operations [11]. Even more directly, it can adversely affect application 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 [11]; 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 decisions. This requirement is true for current grids, and will become more important as resource characteristics become increasingly diverse. We assume the existence of resource availability predictors, which we have studied in previous work [23], and describe further here. Because performance also depends on competing resource load during the lifetime of the application, schedulers must also make use of load monitors and predictors [25], which can be centralized, or can be distributed using various dissemination approaches [12].

In scheduling for performance and reliability, we investigate the effect on both application makespan and the number of wasted operations due to application evictions, which can occur when a resource executing an application 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 application. 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 applications. In this paper, we use our best previously identified availability predictor [23] to investigate the effects of weighing resource speed, load, and reliability in a variety of ways, to decrease makespan and increase application reliability.

¹ This research is supported by NSF Award CNS-0454298.
We also investigate using different approaches to schedule applications with different characteristics. In particular, we propose scheduling checkpointable jobs to consider speed and load more heavily than reliability, because their eviction involves fewer wasted operations when compared with non-checkpointable jobs due to their ability to save state and resume execution elsewhere.

Our simulation-based performance study begins by establishing the inherent performance/reliability scheduling tradeoff for a real world environment. We then introduce and characterize the performance of several different schedulers that consider both reliability and performance, in a variety of ways. We develop and explore the idea of varying the requested prediction duration and show its effect on application execution performance. We investigate the complexity of scheduling for these two competing metrics; we characterize the effect of (i) treating checkpointable jobs differently from "non-checkpointable" jobs, and (ii) varying the checkpointability, length, and number of jobs. Our approach assumes a multi-state availability predictor, which we described previously [23]. We compare it with the only other multi-state availability predictor and scheduler, from Ren et al. [19], and characterize the relative performance and configurability of the two approaches. Our results show that Ren's scheduler causes up to 51% more evicted jobs while simultaneously increasing average job makespan by 18% when compared with our best scheduler.

2. Background

This section describes our previous work, which has led to the contributions described in this paper. In particular, Section 2.1 begins by describing our multi-state availability model, which separates unavailable resources into classes based on why they are unavailable [22], rather than viewing machines as simply available or unavailable. For example, machines that fail unexpectedly are identified separately from those that are unavailable because they are reclaimed by users. We argue that this categorization is important for two reasons. First, applications with varying characteristics will react differently to different types of unavailability. Checkpointable jobs may do better on machines whose users come and go frequently because they can take a checkpoint when the user returns while they may do poorly on machines that become unavailable abruptly, precluding a checkpoint. On the other hand, long-running non-checkpointable jobs may do better to avoid volatile resources that frequently experience any type of unavailability, due to the restarts and lost operations that result from their frequent failures. Section 2.2 therefore identifies the characteristics that influence how applications respond to different kinds of resource unavailability.

A multi-state availability model is also important because it can help predictors be more precise about how they forecast unavailability. Section 2.3 discusses predicting the unavailability states using a variety of approaches. Our best predictors outperform the only other multi-state predictor [19] (which in turn also outperforms other predictors) by approximately 4.6%, a difference that is exaggerated in scheduling results, as we show in this paper.

2.1 Unavailability Types and Multi-State Model

Resources in non-dedicated grids oscillate between being available and unavailable to the grid, based on resource failure characteristics, owner policies, scheduling mechanisms, and application offered load. To motivate our availability states, consider Condor [16], which manages cluster and grid resources. Condor allows individual owners to set their own policies for how and when their resources are utilized. Default policies [1] attempt to minimize Condor's disturbance of local users and processes, whereas customized settings can be more generous in donating resources. We identify five availability states [22]: Available, User Present, CPU Threshold Exceeded, Job Eviction, and Unavailable.

- Available: A machine in this state is currently running with network connectivity, has no user present, and a local CPU load below the CPU threshold, allowing the Grid to use the resource
- CPU Threshold Exceeded: A resource may transition to this state if the local CPU load becomes too high.
- User Present: A resource may transition to the User Present state if the keyboard or mouse is touched.
- Job Eviction: If the resource remains in either of the previous two suspension states too long, or if the machine is shut down, it transitions to the Job Eviction state (and can potentially checkpoint itself).
- Unavailable: If a machine fails or becomes unreachable, it directly transitions to the Unavailable state (checkpointing is impossible).

These states differentiate the types of unavailability. If a job is suspended, and enters the Job Eviction state, we call this a Graceful transition because it may take a checkpoint and migrate to another resource; a transition directly to Unavailable is Ungraceful. Our predictors attempt to forecast the occurrence of these different types of unavailability to help schedulers place applications to exploit the availability characteristics that resources exhibit.

2.2 Grid Application Diversity

Grid applications vary in their ability to tolerate faults. A checkpointable application need not be restarted from the beginning if its host resource transitions gracefully to Unavailable; instead it could take an on-demand checkpoint. Unfortunately, not all jobs are checkpointable [1]. Furthermore, grid applications may complete in a few minutes, or require many hours or even days [3]. Longer jobs will experience more faults, increasing the importance of their varied ability to deal with them. Grid resources will exhibit different characteristics in terms of how long they reside in each availability state, how often they transition between the states, and the states to which they transition. Different applications will behave differently on different resources. If a checkpointable job is suspended and then eventually
gracefully evicted, Condor can checkpoint the job’s current state and resume its execution on another machine. An ungraceful transition requires using the most recent periodic checkpoint, if one exists. A job that is not checkpointable must restart from the beginning, even when gracefully evicted. Some applications without side-effects may be readily replicable, an effective weapon against resource failure. Other applications must run at most once. Scheduling fundamentally different kinds of applications differently from one another could potentially decrease individual application makespan, and increase overall grid throughput and performance.

### 2.3 Multi-State Availability Prediction

We have investigated several different approaches to predict resources’ transition into various unavailability states. Each of them analyzes some period of a resource’s recent history to forecast future availability. Our predictors take a resource, prediction time, and estimated application duration (i.e. "prediction interval") as input, and generate a vector of percentages corresponding to the predictor’s forecast for the likelihood that the resource will next enter each of the availability states (Section 2.1), within the prediction interval. The predictor also forecasts the likelihood that the resource will complete the interval without becoming unavailable.

In considering recent history, different predictors consider either Transitional behavior by counting transitions between our states, or Durational behavior, by summing the total time spent in each state. Our predictors also consider behavior within the most recent N hours, for many values of N, and at similar times of day over the past N days (for various numbers of days). Finally, we consider three different ways of weighing unequally the past behavior within the analyzed history; different predictors weigh more heavily behavior from the same days of the week, from the same times of day, and leading up to the current prediction. Each of these correlates to future availability behavior [22].

Two different predictors emerge from our studies as being the most effective. Our Transitional Day-of-week Equal-weight (TDE) predictor considers transitional behavior for the same prediction interval in the past 16 equally weighted days. For a range of tests, this predictor is more accurate than all others from related work and from all of our other predictor configurations, including the best existing related work predictor [20][21] (which it outperforms by 4.6%). Our predictor that leads to the best scheduling results (for one representative scheduler) is our Transitional Recent-hours Freshness (TRF) predictor. TRF considers transitional behavior from a resource’s past 216 hours (9 days), weighing the most recent (i.e. freshest) behavior most heavily. When used with one sample scheduler, TRF decreases makespan by up to 27%, or operations lost by approximately 30% (with recent improvements in Section 7 of up to 50%) when compared with the existing approach.

Our previous work [23] analyzed the predictors in detail. In this paper, we consider the predictor (TRF) that leads to the best scheduling, and explore the results that a variety of schedulers can achieve with it. Throughout the rest of the paper, the scheduling results we report utilize the same basic scheduling mechanism of resource scoring (with the exceptions of the Ren MTTF, Random and Pseudo-Optimal schedulers). 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. Resource scoring schedulers give each resource a score based on factors that define the scheduler’s placement policy (as described in Sections 4 and 5); the resource with the highest score executes the job. Ties are broken arbitrarily. We focus our analysis on the number of jobs evicted, the number of operations lost as a result, and the average job makespan. (Throughput comparison does not manifest differences in schedulers as well as makespan does.)

### 3. Related Work

Related work resides in two broad categories, namely (i) prediction of resource state (availability and load), and (ii) grid scheduling, especially approaches that consider reliability and availability. We organize this section accordingly, emphasizing in both sections the most closely related project, which also uses multi-state availability prediction, and with which we compare our work most closely later in this paper. [20][21]

#### 3.1 Prediction

Network Weather Service (NWS) [25] uses many linear models to predict host load, and combines them in a mixture-of-experts approach that chooses the best model. The RPS toolkit [10] uses a set of linear host load predictors, including BM(p) (or Sliding Window). Mickens and Noble [17] use variations and combinations of Saturating and History based counters including a hybrid approach, to predict the likelihood of host availability.

Ren et al. [20][21] use empirical host CPU utilization and resource contention traces to develop the only multi-state model (other than ours) for resource availability, and to analyze availability durations by hour of day. A multi-state model considers various types of availability, beyond just “available” or “unavailable.” Ren’s 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. They develop a multi-state predictor that we call Ren N-Day. Our multi-state model and predictor combination differs by capturing user presence on a machine (matching the default Condor resource donation policy), distinguishes between graceful and ungraceful process eviction, and counts transitions differently.

We provide a more detailed review and differentiation of prediction approaches elsewhere [23].

#### 3.2 Grid Scheduling

Most Grid scheduling research attempts to decrease application makespan or to increase throughput. For example, Kondo et al. [14] explore scheduling in a volunteer computing

Fewer projects focus on scheduling for reliability. Kartik and Murphy [13] calculate the optimal set of processor assignments based on expected node failure rates, to maximize the chance of task completion. Qin et al. [18] investigate a greedy approach for scheduling task graphs onto a heterogeneous system to reduce reliability cost and maximize the chance of completion without failure. Similarly, Srivanasan and Jha [24] 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 for performance can be detrimental due to the performance ramifications of failures. Dogan and Ozguner [11] 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. [15] 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. [19] also examine scheduling jobs with their Ren N-day predictor. The Ren MTTF scheduler first calculates each resource’s mean time to failure (MTTF) 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:

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

$CR_i$ 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.

4. Reliability Performance Relationship

This section establishes the inherent tradeoff between scheduling for reliability and for performance for a real world environment. 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 for both performance-based metrics like makespan, and 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") [11].

Our simulation-based experiments that vary the number of jobs, their characteristics, and the schedulers’ scoring techniques, demonstrate this tradeoff. We traced the Notre Dame Condor pool (approximately 600 nodes) for a six month period in 2007 to capture its resources’ multi-state availability and CPU load data. We describe details of the trace elsewhere [22]. We simulate 6000 applications executing on these machines, each utilizing its own recorded availability and load measurements. Our simulation creates and inserts each application at a random time during the six month simulation, such that application injection is uniformly distributed across the six months. The simulation assigns each application a duration (i.e. a number of operations needed for completion); application duration is uniformly distributed between five minutes and 25 hours (the duration is an estimate based on the execution of the application on a resource with an average MIPS speed with no load). 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. In all simulations, 25% of the applications are checkpointable; that is, they can take an on demand checkpoint in five minutes if the user returns or if the local CPU load goes above 50%, leading to eviction and rescheduling. The remaining 75% non-checkpointable jobs need to start over upon eviction.

Each machine’s MIPS score (as defined by Condor’s benchmarking tool), and the load and availability state information contained in the trace, influence the simulated running of the application. During each simulator tick, the simulator calculates the number of operations completed by each working resource, and updates the records for each executing application. If a resource executing an application leaves the Available state (as per the trace), effectively evicting the application, the executing application joins the back of the application queue for rescheduling. The number of operations that remain to be completed for this evicted application depends on the checkpointability of the
application and the type of eviction (Section 2.2). All waiting applications (scheduled or rescheduled) remain queued until they reach the head of the queue, at which point they are rescheduled. During each simulator tick, the scheduler places applications off the head of the queue until the next application cannot be scheduled (e.g. because no resources are available).

To choose from among resources for application execution, schedulers score each available resource according to the following expression:

$$RS_i = (1 - W) \times P_i[\text{COMPLETE}] + W \times (\text{MIPS}_i / \text{MIPS}_{\text{max}}) \times (1 - L_i)$$

$P_i[\text{COMPLETE}]$ is resource $i$’s predicted probability of completing the job interval without failure, according to the TRF predictor, $\text{MIPS}_i$ is the resource’s processor speed, $\text{MIPS}_{\text{max}}$ is the highest processor speed of all resources (for normalization) and $L_i$ is the resource’s current processor load. Reliability influences completion probability $P_i[\text{COMPLETE}]$, performance influences $(\text{MIPS}_i / \text{MIPS}_{\text{max}}) \times (1 - L_i)$, and the Tradeoff Weight ($W$) determines which more heavily influences the resource’s overall score.

Figure 1 depicts 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 but 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.

5. Prediction Based Scheduling Techniques

This section explores in more detail a wider range of resource ranking strategies that consider resource performance and reliability. Section 5.1 studies the effect of considering various combinations of CPU speed, load, and reliability; the section also introduces the idea of scheduling checkpointable jobs differently from non-checkpointable jobs. We then describe how the best performing scheduler from Section 5.1 behaves as we vary the length of the interval for which the predictor forecasts (Section 5.2).

5.1 Scoring Technique Performance Comparison

We consider a range of resource scoring approaches, which utilize some or all of the following factors.

- CPU Speed ($\text{MIPS}$): the resource’s Condor MIPS rating
- Current Load ($L$): The machine utilization information at scheduling time.
- Completion Probability ($P[\text{COMPLETE}]$): The predicted probability of completing the projected job execution time without becoming unavailable. This is one minus the sum of the probabilities of the three unavailability states.
- Ungraceful Eviction Probability ($P[\text{UNGRACE}]$): The predicted probability of exiting job execution directly to unavailable (with no chance for a checkpoint).

![Figure 1: As schedulers consider performance more than reliability (by increasing Tradeoff Weight $W$), makespan decreases, but the number of evictions increases.](image1)

![Figure 2: The resource scoring decision’s effectiveness depends on the length of jobs. For longer jobs, schedulers should emphasize speed over predicted reliability.](image2)
Checkpointability: Whether the job could take an on-demand checkpoint before being evicted from a machine (CKPT) or not (NON_CKPT)

Considering whether or not the scoring system incorporates each of the first three criteria defines eight different kinds of scoring, ranging from considering none of the criteria (Random scheduling), to including them all. Then, if checkpointable jobs are scheduled differently from non-checkpointable jobs, many more possible approaches emerge. There is also the matter of how to incorporate each factor into the score. Based on intuition (some combinations make more sense than others) and background experimental results, we focus on the following resource scoring approaches and provide the formula for computing each resource’s score, $R_S_i$

- **S0**: $MIPS_i \times (1 - L_i)$
- **S1**: $P_{[COMPLETE]}$
- **S2**: $MIPS_i \times P_{[COMPLETE]}$
- **S3**: $MIPS_i \times (1 - L_i) \times P_{[COMPLETE]}$
- **S7**: CKPT $MIPS_i \times (1 - L_i) \times (1 - P_{[UNGRACE]})$
- **S9**: CKPT $MIPS_i \times (1 - L_i) \times P_{[COMPLETE]}$

S0, S1, S2, and S3 schedule checkpointable jobs the same way they schedule non-checkpointable jobs with S0 considering speed, S1 considering reliability, and S2 and S3 considering both. However, the fact that checkpointable jobs react differently to resource failure suggests that scheduling them differently could improve overall grid performance. We therefore also consider S7 and S9.

We ran simulations for the same environment and setup as in Section 4, but varied the maximum job length from 12 to 120 hours. $P_{[COMPLETE]}$ (an availability prediction) is made by TRF and used as input to the resource scoring decisions that utilize it. The results we obtained and present in the remainder of the paper are based on simulating each scenario once. However, we have verified the consistency and repeatability of the results through repetition.

Figure 2 plots average makespan vs. maximum job length, and evictions vs. maximum job length for all of our proposed resource scoring decisions as percentage difference versus S0. We use percentages for comparison throughout the paper. We

---

Figure 3: Scoring decision effectiveness depends on job checkpointability. Schedulers that treat checkpointable jobs differently than non-checkpointable jobs (S7 and S9) suffer more evictions, but mostly for checkpointable jobs; therefore, makespan remains low.
plot the percentage difference of one scheduler versus a chosen baseline scheduler for a particular metric (e.g. operations lost or evictions). We use percentages to emphasize the differences in scheduling quality and to facilitate visual comparison of performance, given the large range of $y$ values. The text includes the magnitude of the results (e.g. the actual number of evictions) in several places.

S1, which considers only reliability, performs significantly worse than all the others in terms of makespan for all maximum job lengths, but also for number of evictions for longer jobs. This demonstrates that schedulers must consider resource performance when scheduling. The figure also shows that makespans are relatively similar for the other scheduling approaches, but that S3 does the best job of avoiding evictions. For example, for jobs up to 84 hours, S3 obtains 5417 evictions whereas S1, S7 and S9 obtain approximately 6450 evictions (19% more). For a shorter job length of 6 hours, S3 causes 860 evictions and S7 and S9 cause about 1070 evictions increasing the disparity to 24%. These results demonstrate that schedulers must consider both reliability and performance while scheduling, to simultaneously produce fewer evictions and shorter job makespan.

Figure 2's results are for 25% checkpointable jobs. When that percentage varies, the value of considering reliability in the scoring function changes, as does the value of scoring checkpointable and non-checkpointable jobs differently.

5.2 Prediction Duration

In tests described so far, the scheduler asks the predictor about behavior over a prediction interval that is intended to match application runtime. A prediction for the next N hours may not necessarily reflect the best information for scheduling an N hour job. This section investigates the effect of scheduling an N hour job using predictions made for the next $(M \times N)$ hours, where $M$ is the “interval multiplier.” We explore results for $M$ in {0.25, 0.5, 1, 2, 3}, plot 0.25 as a baseline, and investigate a two “hybrid” multipliers. The “Comparison” (Comp or TRF-Comp) scheduler uses $M=3$ for jobs less than 28 hours, and $M=0.25$ for longer jobs. The “Performance” (Perf or TRF-Perf) scheduler ignores reliability on jobs less than 28 hours (instead selecting the fastest, least loaded resources); for longer jobs, it uses $M=0.25$.

Figure 4 shows that larger multipliers perform better for jobs up to 28 hours in duration. However for longer jobs, making predictions for intervals longer than the application runtime increases the number of evictions and the average job makespan. In fact, the longer the job, the smaller the optimal multiplier. This observation motivated the design of the hybrid scheduler, which does well in keeping evictions low, through 80-hour jobs, and makespan low, especially for longer...
jobs. For example, for short jobs of length 6 hours, M=0.25 produces 1173 evictions, M=1 produces 878 evictions and M=3 produces 754 evictions. For 84 hour jobs, M=0.25 has 80,118 evictions, M=1 has 84,662 and M=3 has 87,433. These figures support the conclusion that the longer the job, the shorter the optimal prediction length multiplier should be.

6. Multi-State Prediction Based Scheduling

This section compares our predictor and scheduler with Ren’s multi-state prediction based scheduler [19], in terms of the ability to tradeoff reliability and performance. To isolate the effect of the predictor from that of the scheduler, we also compare our predictor and scheduler combination with Ren’s predictor using our scheduler.

6.1 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, while simultaneously varying TRF’s interval multiplier (Section 5.2) for comparison. Both of these parameters allow each scheduler to trade off reliability and performance.

Figure 5 shows the average makespan and number of evictions obtained 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 1500 evictions, TRF has average makespan that is 27% lower. And for makespan of 13 hours, TRF has 52% fewer evictions.

6.2 Predictor Quality vs. Scheduling Result

To better understand how predictors affect scheduling, we test our scheduler with both our predictor and with Ren N-Day [20][21] for a variety of interval multipliers (Prediction Length Weights). We simulate 6000 jobs ranging from 5 minutes to 25 hours in duration, 25% of which are checkpointable. We vary N in the Ren predictor, and prediction length weights (i.e. multipliers, as outlined in Section 5.2) between 1 and 7, and follow the simulation setup.

Figure 5: TRF and Ren MTTF show similar trends for both makespan and evictions. For any particular makespan, TRF causes fewer evictions, and for any number of evictions, TRF achieves improved makespan.

Figure 6: TRF causes fewer evictions than Ren (coupled with our scheduler) for all Prediction Length Weights. TRF also produces a shorter average job makespan than all Ren predictors, except the one day predictor (which produces 45% more evictions).
explained in Section 4.

Figure 6 illustrates the percentage difference in makespan, vs. TRF. For all multiplication factors, TRF produces the fewest job evictions. Both predictors produce the fewest evictions with the 7x multiplier; Ren 2-Day produces 1,340 evictions and TRF produces 1,103 (21% fewer). For the 1x multiplier, Ren 12-Day (1,534 evictions) comes closest to TRF’s 1,420 evictions (within 8%). For makespan, Ren 1-Day beats TRF by at most 18% lower makespan. However, Ren 1-Day sacrifices reliability and produces about 45% more evictions. Thus, TRF reduces the number of evicted jobs and can obtain the highest job reliability. Ren 1-Day coupled with our scheduler can obtain a shorter makespan than TRF, but only with a large loss in job reliability.

7. Performance Comparison

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 six month Notre Dame trace. The jobs range from five minutes to the job length indicated on the x-axis.

Figure 7’s top two graphs compare the TRF schedulers to Ren-MTTF (with a variety 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, vs. TRF-Comp. The History and Sliding Window predictors utilize the Comp scheduler 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 hours 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 hours, TRF-Comp achieves at least 15% fewer evictions when compared with the most reliable scheduler, Ren MTTF-1; average job makespan simultaneously decreases by 20% (27.3 hours versus 32.9 hours). TRF-Comp also decreases the number of evictions by at least 57% compared with all other schedulers, for jobs up to 6 hours 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+ hours.

The bottom two graphs in Figure 7 compare TRF-Comp and TRF-Perf with non-prediction-based scheduling approaches.
including a Pseudo Optimal scheduler which selects the available resource that will execute the application in the smallest execution time, without failure, based on omnipotent future knowledge of resource availability. When all machines would fail before completing the application, the Pseudo Optimal Scheduler chooses the fastest available resource in terms of MIPS speed. For average job makespan, TRF-Comp follows the non-optimal schedulers and produces the fewest evictions for all job lengths, by at least 110%.

We’ve also performed this same analysis through varying load. Due to space constraints we are unable to include these results but across a large variety loads, TRF-Comp produces superior results a

8. Summary

The volatility of resources can have important consequences on executing applications. In particular, the type of resource unavailability an application experiences coupled with its ability to checkpoint can lead to different performance characteristics. We have shown that prediction-based schedulers aware of the various types of unavailability can take advantage of this information by making better job placement decisions. Our TRF-Comp scheduler outperforms other traditional schedulers, and other schedulers that trade off reliability for performance, including the only other scheduler based on multi-state availability prediction. We show reduced makespan and fewer evictions, for a variety of scenarios.

9. References