A Performance Comparison of Resource Allocation Policies in Distributed Computing Environments with Random Failures

Bhavesh Khemka[1], **Anthony A. Maciejewski**[1], **and Howard Jay Siegel**[1,2]
[1]Department of Electrical and Computer Engineering, [2]Department of Computer Science,
Colorado State University, Fort Collins, Colorado, USA

Abstract— *The problem of efficiently assigning tasks to machines in heterogeneous computing environments with uncertainty in the availability of the compute resources is a challenging one. Previous research has looked at designing heuristics to maximize the total reward earned by completing tasks in an environment where compute nodes may randomly fail. The rewards associated with the tasks are earned if they are successfully executed before their deadlines. The goal of the resource allocation policies is to maximize the cumulative reward earned. We use heuristics from the literature and improved versions of some of the heuristics to perform the resource allocation decisions. We conduct extensive experiments to compare the performance of these heuristics in a variety of simulation environments. The goal of the study is to be able to recommend a heuristic to use based on the system environment. Our experiments show that different heuristics perform the best in different environments.*

Keywords: mapping, resource allocation, fault-tolerant, heterogeneity, rewards, deadlines

1. Introduction

Distributed computing is currently used to solve a host of problems where different tasks are mapped to separate machines for execution. These environments may be heterogeneous, which means different tasks may have varied execution times on the different machines. This makes it difficult to assign tasks to machines to optimize for a given performance metric. The process of allocating tasks to machines for execution is commonly referred to in the literature as "resource allocation" or "mapping." The mapping and scheduling problem has been known to be NP-Complete [1], and therefore one must use heuristics to get a solution to this problem. It is common for failures to randomly occur in the compute resources of these large-scale distributed systems. As a result, it becomes even more difficult to make resource allocation decisions while being aware of such failures. In this study, the goal of our resource allocation procedures is to maximize the total reward earned while executing the tasks in an environment where the machines may randomly fail.

This work was supported by the National Science Foundation under grant number CNS-0905399, and by the Colorado State University George T. Abell Endowment.

Fault-tolerance in distributed computing environments has been extensively studied. Check-pointing tasks to avoid restarting them from the beginning, in case they fail, is a common method used to alleviate the damage caused by the failure of resources [2], [3], [4], [5]. Replicating tasks is another method used to improve the reliability of the system [6], [4], [7], [8].

Shestak et al. [9] addressed the problem of making resource allocation decisions while being aware of the mean fail rates of the compute resources. To model a harsh environment the machines were made to have high probabilities of failure. They assumed that a bag of tasks is available and ready for execution. Each of those tasks has a reward and a deadline associated with it. The reward of a task is earned if the task is successfully completed by its deadline. If a machine fails while executing the task, the task is returned to the batch and may be re-mapped to another machine for execution. The goal of the resource allocation heuristics is to maximize the total reward that can be earned by executing the tasks before their respective deadlines expire.

The contributions of this paper are: (a) an enhancement in the prediction mechanism for some of the heuristics from the literature [9], and (b) a study of the performance of different heuristics on various environments that differ in their types of heterogeneities, rates for task executions and machine failures, and task re-mapping policies. We compare the relative performance of the heuristics in each of these simulated environments, and recommend the heuristic that one should choose to obtain the highest reward for each environment.

2. Problem Statement

The environment we model in this study and the goal of our resource allocation procedures are based on the work of Shestak et al. [9]. Each task has an associated reward that is earned if its computation is successfully completed by its deadline. The compute resources are assumed to have high probabilities of failure. When a machine failure occurs while executing a task, the task is returned to the batch and is eligible to be mapped to another machine. The task may fail multiple times, and it can be continually remapped to

constant	**mach A**	**mach B**	**mach C**
task class 1	4	4	4
task class 2	4	4	4
task class 3	4	4	4

column-varying	**mach A**	**mach B**	**mach C**
task class 1	5	2	4
task class 2	5	2	4
task class 3	5	2	4

inconsistent	**mach A**	**mach B**	**mach C**
task class 1	6	5	7
task class 2	9	3	4
task class 3	2	10	1

task-mach-consistent	**mach A**	**mach B**	**mach C**
task class 1	1	2	7
task class 2	3	4	9
task class 3	5	6	10

Fig. 1: Sample Estimated Time to Compute (ETC) matrices modelling different types of heterogeneity in an environment with three task classes and three machines

machines as long as its deadline has not expired. The goal of the resource allocation policies will be to map tasks to machines to maximize the total reward earned from all the tasks. In this study, we say that a heuristic is "robust" [10] if it can earn reward in an environment with uncertainties in task execution times and machine failure times. The higher the reward a heuristic earns, the more robust it is to such uncertainties.

We work with a bag of tasks that was available at the start of the simulation. Whether or not failed tasks are allowed to come back for mapping events may be a policy decision. Therefore, we consider both cases (allowing and not allowing failed tasks to re-map), and simulate both environments for experimentation purposes. We also model various types of heterogeneity of the computing environment, and gauge the performance of the various resource allocation procedures.

3. Environment Modeling

3.1 Modeling the Task Execution Times

An Estimated Time to Compute (ETC) matrix is used to model the execution time characteristics of the various tasks in the heterogeneous system. We use task classes to group together tasks that have similar execution time characteristics. Each entry t_{ij} in the ETC matrix gives the mean execution time of tasks of class i on machine j of our heterogeneous suite. The actual execution time of the tasks are modeled using exponential distributions with the means obtained from the entries of the ETC matrix. We use exponential distributions to model task completion times, based on the tests conducted at Ricoh InfoPrint [11]. For simulation purposes, we create and use synthetic workloads, but in real-world environments one could build such a matrix based on historical data.

We model and gauge the performance of various resource allocation policies under different types of ETC matrices. The different ETC matrices are used to model various types of heterogeneity of computing systems. We model four types of ETC matrices. Sample 3 x 3 ETC matrices for each of these types are shown in Figure 1. We model homogeneous workloads and homogeneous compute resources by having a fixed value for all the entries in the ETC matrix. We call this type of ETC matrix *constant*. We model another environment wherein the workload can be considered homogeneous, but the compute resources can be considered to have different computational capabilities. We model such an environment by having unique values for each of the columns of the ETC matrix, and refer to this matrix as *column-varying*. A completely heterogeneous environment is modeled by having random values for each cell in the ETC matrix. In an environment, modeled by such a matrix, it is possible (and likely) for a machine to be better than another machine for a particular task class and worse for another. Such a matrix is referred to as *inconsistent*. If we independently sort the elements within each of the rows of such an *inconsistent* ETC, and then independently sort the entries in each of the columns, we obtain what we call a *task-mach-consistent* matrix. In such an environment, if a machine executes a task faster than another machine, then it will do so for all tasks. Similarly, if a task executes faster than another task on a single machine, then it will do so on all machines. This type of matrix is well suited for some of the heuristics discussed later.

3.2 Modeling the Machine Failures

It has been shown in the literature that exponential distributions can be used to stochastically model hardware failures [12], [13]. In our environment, each machine has an exponential distribution associated with it to model the probability of failure, and for each machine j, we have a failure rate represented by λ_j.

Shestak et al. [9] have shown how a distribution of processor availability can be obtained using the average execution time of tasks on a machine j (referred to as t_j^{av}), and the failure rate of the machine λ_j. This probability mass function consists of as many pulses as there are machines in the environment. The distribution gives the relative probability of each machine to become available for a mapping event. Machines become available for a mapping event under two scenarios: if they have successfully completed a task, or if they have encountered a failure. The distribution sorts the machines in an ascending order of their "quality." A machine has better "quality" if it has a lower value for $\lambda_j t_j^{av}$. Therefore, a machine that has a lower failure rate and/or a lower value for mean execution time, will be considered better. The Cumulative Distribution Function (CDF) of this probability distribution will be used to guide resource allocation decisions by some heuristics.

4. Resource Allocation Policies

4.1 Overview

In our environment, a mapping decision (deciding which machine to map a task to) is made whenever a machine becomes available for executing a task. Machines become available in two cases: when they successfully complete a task that was assigned to them, or when they have encountered a failure. We make an assumption that failed machines get instantly repaired and are available for executing tasks immediately. We use heuristics from the literature [9] as well as modifications to these heuristics to make mapping decisions.

4.2 Heuristics from the Literature

4.2.1 Reward Heuristics:

There are two heuristics that directly try to optimize the reward [9]. They are the *Reward* heuristic and *Expected Reward* heuristic. In *Reward*, whenever a mapping event occurs, the task that has the highest value for reward is assigned to the machine that just became available. In *Expected Reward*, when a machine becomes available for a mapping event, the task with the highest value for expected reward is assigned to it.

For a task i, $\underline{P_i(t)}$ is the probability (computed at time t) of successfully completing task i through multiple assignments before its deadline expires. The expected reward for a task i is given by the product $r_i P_i(t)$, where r_i is the reward that this task can earn if completed successfully. For any machine j, we represent by $\underline{p_j}$ the probability of machine j being available for an assignment. Also, the term $\underline{V_i(t)}$ is the estimated number of reassignments that task i may undergo starting at time t up to its deadline. The derivations of these terms are shown in [9]. $P_i(t)$ is calculated using the equation shown below.

$$P_i(t) = 1 - \left(\sum_{j=1}^{M} p_j (1 - e^{-\lambda_j t_{ij}}) \right)^{V_i(t)} \qquad (1)$$

The term $(1 - e^{-\lambda_j t_{ij}})$ gives the probability of task i failing on machine j. This factor is weighed with the probability of machine j being available for an assignment (p_j), and therefore the weighted sum, $\sum_{j=1}^{M} p_j(1 - e^{-\lambda_j t_{ij}})$, gives the probability of failure when task i is mapped to a machine. Therefore, Equation 1 represents the probability that task i will successfully complete before its deadline, even through multiple assignments.

4.2.2 Matching Heuristics:

There are two heuristics in [9] that use the concepts of the Derman-Lieberman-Ross (DLR) Theorem [14] to guide mapping decisions. We call them the *Matching* heuristic and *Expected Matching* heuristic. A brief overview of the DLR theorem is given below, followed by the *Matching* heuristics that are implemented using the DLR concept.

The DLR theorem [14] provides an algorithm for optimally assigning a set of available workers to incoming jobs. Each incoming job is assumed to have a reward value associated with it. Each worker is assumed to have a probability (that represents the quality and skill of the worker), with which the reward earned for a job is scaled. It is also assumed that one has the distribution from which the reward values for all the incoming tasks are sampled from. Using the distribution of the reward values of the incoming tasks, and the notion of the skill of the workers (determined by their probabilities), the DLR method describes an algorithm that maps high reward tasks to better skilled workers and low reward tasks to lesser skilled workers. The distribution that dictates the reward values of the incoming jobs is vital to making these decisions.

The *Matching* heuristics [9] try to implement the DLR concept within the resource allocation problem. In our environment, machines become available for mapping events, and a task needs to be assigned to them. This is analogous to jobs coming in and looking for a worker that can be assigned to them. The distribution described in Section 3.2 is used to describe the quality and the likelihood of the incoming machine, analogous to the distribution that governs the likelihood of various reward values for the incoming job. The only other factor that needs to be accounted for is the ranking of the tasks, analogous to the ranking of the workers. It is in this aspect that the *Matching* and the *Expected Matching* heuristics differ. In *Matching*, the tasks are sorted based on their reward values. In *Expected Matching* the tasks are sorted based on their value of expected reward. As before, expected reward of a task i is given by the product $r_i P_i(t)$.

4.3 Modifications to Heuristics

We modify Equation 1 to incorporate the knowledge of the machine that just became available for a mapping event. Let us call the machine that just became available to be machine J. The probability that this machine will become available p_J will be 1, and by a similar logic $p_j = 0, \forall j \neq J$. Therefore, the summation term for this mapping event will reduce to $(1 - e^{-\lambda_J t_{iJ}})$. We know that this counts as an assignment for task i, and therefore we extract the term

$(1 - e^{-\lambda_J t_{iJ}})$ out, and reduce the count of the number of reassignments of task i (denoted by $V_i(t)$) by one. This gives us our new equation for $P_i(t)$.

$$P_i(t) = 1 - (1 - e^{-\lambda_J t_{iJ}}) \times \left(\sum_{j=1}^{M} p_j (1 - e^{-\lambda_j t_{ij}}) \right)^{V_i(t)-1} \quad (2)$$

We create the *Latest Expected Reward* and the *Latest Expected Macthing* heuristics that are similar to the *Expected Reward* and *Expected Matching* heuristics, but with the difference that they use Equation 2 for their expression of $P_i(t)$ instead of Equation 1. This is to denote the fact that they use the latest information to compute $P_i(t)$.

For experimentation purposes, we also model an environment where tasks are not allowed to come back when the machine they were assigned to failed. In such an environment, the value of $P_i(t)$ is simply calculated using the equation given below.

$$P_i(t) = e^{-\lambda_J t_{iJ}} \quad (3)$$

5. Simulation Setup

In this study, the workload that we are modeling consists of independent tasks, i.e., no communication is required between the individual tasks, and there are no precedence constraints. Each task is sequential (i.e., not decomposable into parallel parts). It is also assumed that each machine can handle only one task at a time (no multitasking).

We modeled different ETC matrices, as described in Section 3.1. The entries of the ETC matrix represent the mean execution time values of the different task classes on the different machines. We modeled two types of environments; the *narrow* and the *broad* environments. For the *narrow* environment, the entries of the ETC matrix were uniformly picked at random from the range [0.5, 4.0], whereas for the *broad* environment they were uniformly picked at random from the range [0.5, 9.5].

The values of the mean times to failure for the machines were also randomly picked from a range depending on the environment that was being modeled. For the *narrow* environment, the range was [0.6, 1.0], whereas, for the *broad* environment, the range was [0.6, 2.375]. Therefore, the *broad* environment has wider ranges for the possible values of the execution times of the tasks and the times between failure for the machines in comparison to the *narrow* environment.

Similar to the environment studied by Shestak et al. [9], we model an environment with 200 tasks, six machines, and five task classes. Each task is randomly assigned to one of the five task classes. The ETC matrix has task classes along its rows, and machines along its columns, and therefore the size of the ETC matrix is 5 x 6. For each task, the reward value was assigned by randomly picking an integer in the range [1, 100]. All the tasks in a task class have a common deadline. The deadline for a task class was set to six times the longest execution time of this task class across the machines.

There were three main parameters that we varied to alter the environment being modeled. The first was whether or not we allow failed tasks to return back to the batch for further remapping. The second was whether we use the *narrow* or *broad* environment. The final parameter was the type of ETC matrix used. We modeled four types of ETC matrices: *constant*, *column-varying*, *inconsistent*, and *task-mach-consistent*. We ran tests with all combinations of these various parameters.

6. Experimental Results and Analysis

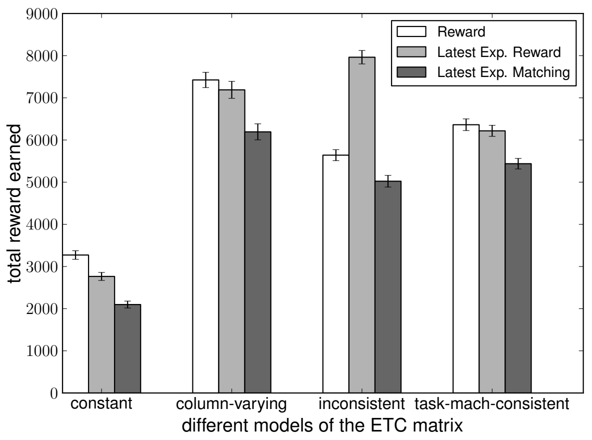
The goal of this study is to compare and evaluate the performance of the heuristics under a variety of scenarios to be able to choose which heuristic to use for different environments. Therefore, it serves to only compare the relative performance of the heuristics with each other under the various scenarios, as opposed to comparing the absolute performance of a heuristic across the scenarios. It would also be inaccurate to make such a comparison, because the different scenarios have different execution time characteristics and different handling methods for dropped tasks. These can result in different values for the total reward earned by the same heuristic under these varied environments.

For each simulation case, 100 trials were performed and the results were averaged and 95% confidence intervals were calculated. For each trial, new values were used for the following: the entries of the ETC matrix, reward values of the tasks, and fail rates of the machines.

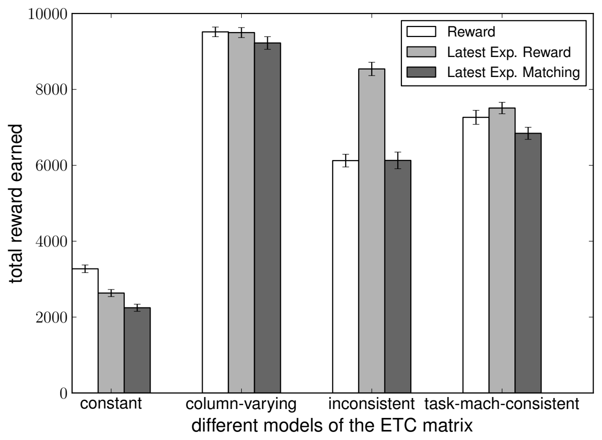
We observed that the *Expected Reward* heuristic did not perform better than the *Latest Expected Reward* heuristic for any of the test cases. The *Latest Expected Reward* heuristic is able to use the most recent information to its advantage while calculating the probability of a task successfully completing. The significant benefit earned by using Equation 2 as opposed to Equation 1 comes from the fact that we are able to factor out the current machine from the summation, thus avoiding distorting the probability of failure for this first mapping event.

From our experiments, we also observed that the *Latest Expected Matching* heuristic performed better than both the *Matching* and the *Expected Matching* heuristics. Therefore, in Figure 2 we only show the results for the *Reward*, *Latest Expected Reward*, and *Latest Expected Matching* heuristics. Each of these three heuristics perform better than the other two heuristics for at least some of the environments.

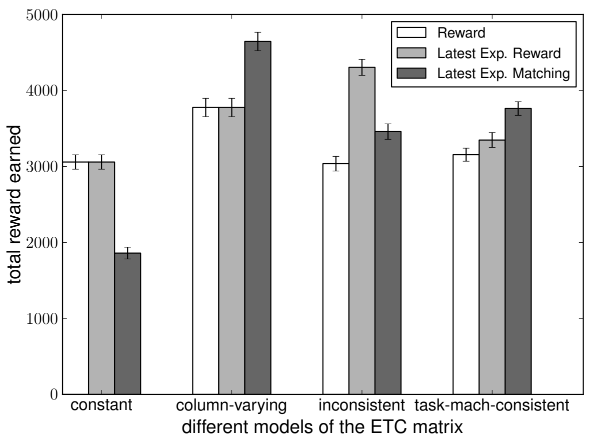
Figure 2 shows the results for the *Reward*, *Latest Expected Reward*, and *Latest Expected Matching* heuristics for the various types of environments that we modeled. Figures 2(a) and 2(b) show results from the case where re-mapping of failed tasks is allowed. Figures 2(c) and 2(d) shows



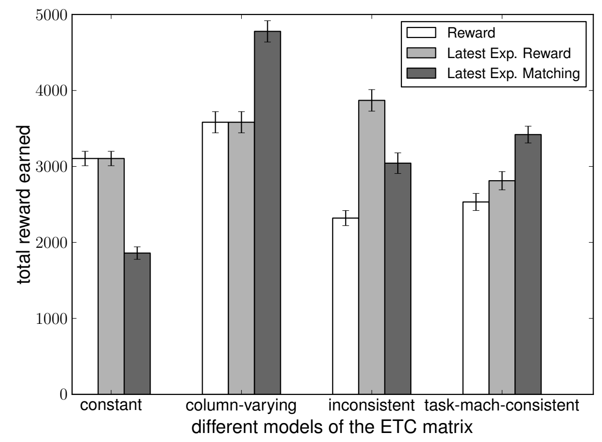
(a)



(b)



(c)



(d)

Fig. 2: Results showing the total reward earned by the *Reward*, *Latest Expected Reward*, and the *Latest Expected Matching* heuristics in (a) the *narrow* environment with re-mapping allowed, (b) the *broad* environment with re-mapping allowed, (c) the *narrow* environment with no remapping, and (d) the *broad* environment with no remapping. The results are averaged over 100 trials and the error bars show 95% confidence intervals.

results with the case where each task can be mapped to a machine only once. Figures 2(a) and 2(c) have results from the *narrow* environment. This determines the ranges of the mean execution times of the tasks and the mean fail times of the machines. Figures 2(b) and 2(d) show results for the case with the *broad* environment. For each of these charts, along the independent axis are the results of the three heuristics for the various types of ETC matrices that we modeled, i.e., *constant*, *column-varying*, *inconsistent*, and *task-mach-consistent*. It is worth noting that our goal will be to compare the relative performance of the three heuristics for any given scenario (i.e., comparing the bars within the set of three) as opposed to comparing the results across scenarios.

The *Reward* heuristic performs the best among the three heuristics when we have a completely homogeneous environment (i.e., modeled by the *constant* ETC matrix) irrespective of whether or not tasks are allowed to come back, and whether we use the *narrow* or the *broad* environment. Note that in the case when tasks are not allowed to come back (Figures 2(c) and 2(d)) and when the *constant* or the *column-varying* ETC matrices are used, the *Latest Expected Reward* and the *Reward* heuristic have equal results. On average, the total reward earned by the *Reward* heuristic was at least a 45% improvement over that earned by the *Latest Expected Matching* heuristic when the *constant* ETC matrix was used. This can be attributed to the fact that the *Reward* heuristic directly optimizes for reward, and in an environment where all tasks perform similarly across all machines, the benefits obtained by performing a matching of tasks and machines is reduced.

In cases where the environment is heterogeneous (i.e., models of the ETC matrix other than *constant*), the *Reward* heuristic beats the *Latest Expect Matching* heuristic when failed tasks are allowed to come back for mapping events (Figures 2(a) and 2(b)). This trend is reversed when failed tasks are not allowed to come back (Figures 2(c) and 2(d)). The *Latest Expected Matching* always does better than the *Reward* heuristic (except the case with the *constant* ETC) when failed tasks are not allowed to come back for re-mapping. Also, in general the relative performance of the *Latest Expected Matching* heuristic is better in such environments. Unlike the other heuristics, the *Latest Expected Matching* heuristic, tries to match "better" tasks (those that have a higher value for expected reward) to "better" machines (those that execute faster, and fail less often). This is particularly helpful when re-mapping is not allowed, because in this case, each task only gets one chance to be mapped to a machine. The *Latest Expected Matching* heuristic performs the best in this case because it holds on to the better tasks until a good machine comes in for a mapping event. The *Reward* (or the *Latest Expected Reward*) heuristic simply assign the task with the highest reward (or expected reward) to the next available machine.

The *Latest Expected Matching* heuristic performs better than the *Latest Expected Reward* heuristic when the environment uses the *column-varying* ETC matrix or the *task-mach-consistent* ETC matrix and when failed tasks are not allowed to re-map. This is because in such environments it becomes easier for the *Latest Expected Matching* heuristic to be able to designate some machines as being better than the others. This helps the heuristic rank the machines in terms of their "quality" and as a result provides better mapping decisions. Moreover, when we use the *broad* environment as opposed to the *narrow* environment, the relative performance benefit of this heuristic increases. This is because there is more variance in the performance of the machines and the *Latest Expected Matching* heuristic is able to use that to its advantage as it makes its decisions by ranking machines in terms of their goodness. The more the difference between the performance of the "good" and "bad" quality machines, the more the benefit of using the *Latest Expected Matching* heuristic.

When we use an *inconsistent* ETC matrix it becomes hard for the *Latest Expected Matching* heuristic to rank the machines. This is because in a highly inconsistent heterogeneous environment, a machine may perform better than another machine for one task, but may perform worse for another task. This makes it harder for the *Latest Expected Matching* algorithm to be able to rank the machines in a global manner in terms of their "quality." Therefore, we see that the *Latest Expected Reward* heuristic always performs better when we model the environment with an *inconsistent* matrix. It performs much better than the *Reward* heuristic in all *inconsistent* ETC matrix cases, because it calculates the expected reward (viz. the probability of earning some reward amount) by looking at the execution time values of the tasks. The *Reward* heuristic fails to look at the heterogeneity of the system while making its mapping decisions, and therefore performs poorly.

In summary, the *Latest Expected Matching* heuristic performs the best when the environment being modeled is similar to the environment of the DLR theorem [14] (on which this heuristic is based), i.e., re-mapping of tasks is not allowed, and the machines can be ranked clearly in terms of their performance (modeled by *column-varying* and *task-mach-consistent* types of ETC matrices). The *Reward* heuristic performs the best when the environment is completely homogeneous (modeled by the *constant* type of ETC matrix), because the execution times are the same for all tasks and machines. The *Latest Expected Reward* heuristic performs the best compared to the other heuristics in a highly heterogeneous environment (modeled by the *inconsistent* type of ETC matrix) because it optimizes for not just the reward value but also the likelihood of earning that reward.

7. Related Work

The scheduling problem has been widely studied in heterogeneous computing environments (eg., [15], [16], [17]). It is important to make the resource allocations be fault tolerant, especially in distributed and grid computing environments. Various techniques have been used to cope with the ill-effects of failures of compute resources. Checkpointing and rollback-recovery are common techniques used to avoid having to restart failed tasks from the beginning (e.g., [2], [3], [4], [5]) (as mentioned in Section 1). Another method used to improve the reliability of the system, in terms of increasing the chances of completing tasks, is to run replicas of the tasks on multiple compute resources (e.g., [6], [4], [7], [8]).

Shestak et al. [9] addressed the problem of maximizing the reward earned by the tasks in an environment where the compute nodes may randomly fail. Their work used the concepts of a theorem introduced by Derman et al. [14]. There have been other works on scheduling that look at maximizing reward earned by the tasks [18], but they do not model environments where the machines tend to fail. Our study builds on the work done in [9] to perform a comparative study of the performance of the heuristics under a variety of system environments.

8. Conclusions and Future Work

The goal of this study was to be able to model and characterize various system environments and gauge the relative performance of fault-tolerant heuristics in these environments. This study extends the work of Shestak et al. [9] by addressing a similar problem, but performing extensive tests on a wide-range of environments. We also modified and improved the prediction mechanism of the *Expected Reward* and the *Expected Matching* heuristics by using the latest information we have about the system. We simulated a variety of environments by changing different attributes associated with the environment. Our results show that the *Reward*, *Latest Expected Reward*, and the *Latest Expected Matching* heuristics have different strengths and weaknesses, therefore performing better or worse depending on the environment.

One direction for future work is introducing a delay before a failed machine returns for a mapping event. This will help us to more closely model a realistic environment. Also, we may try to modify the *Latest Expected Matching* heuristic to make it more heterogeneity-aware. It would also be interesting to modify the workload to have tasks whose reward values degrade with time, instead of having a fixed reward value until a hard deadline.

Acknowledgments: *The authors thank Ryan Friese and Mark Oxley for their valuable comments.*

References

[1] M. R. Gary and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[2] D. Manivannan, "Checkpointing and rollback recovery in distributed systems: existing solutions, open issues and proposed solutions," in *Proceedings of the 12th International Conference on Systems*. World Scientific and Engineering Academy and Society (WSEAS), 2008, pp. 569–574.

[3] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 1, pp. 23–31, Jan. 1987.

[4] Y. Zhang, A. Mandal, C. Koelbel, and K. Cooper, "Combined fault tolerance and scheduling techniques for workflow applications on computational grids," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, 2009, pp. 244–251.

[5] B. Nazir, K. Qureshi, and P. Manuel, "Adaptive checkpointing strategy to tolerate faults in economy based grid," *The Journal of Supercomputing*, vol. 50, pp. 1–18, 2009.

[6] A. Litke, D. Skoutas, K. Tserpes, and T. Varvarigou, "Efficient task replication and management for adaptive fault tolerance in mobile grid environments," *Future Generation Computer Systems*, vol. 23, no. 2, pp. 163–178, Feb. 2007.

[7] Y. Oh and S. Son, "Scheduling real-time tasks for dependability," *The Journal of the Operational Research Society*, vol. 48, no. 6, pp. 629–639, June 1997.

[8] Q. Zheng, B. Veeravalli, and C.-K. Tham, "On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs," *IEEE Transactions on Computers*, vol. 58, pp. 380–393, Mar. 2009.

[9] V. Shestak, E. K. P. Chong, A. A. Maciejewski, and H. J. Siegel, "Probabilistic resource allocation in heterogeneous distributed systems with random failures," *Journal of Parallel and Distributed Computing*, accepted to appear.

[10] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 7, pp. 630–641, jul 2004.

[11] InfoPrint. (accessed Mar. 2012). [Online]. Available: http://www.infoprint.com/internet/ipww.nsf/vwWebPublished/print-infoprint-5000-en

[12] C. E. Ebeling, *Introduction to Reliability and Maintainability Engineering*. Waveland Pr Inc, 2005.

[13] M. Rausand and A. Høyland, *System Reliability Theory: Models, Statistical Methods, and Applications*. Wiley-Interscience, 2008.

[14] C. Derman, G. J. Lieberman, and S. M. Ross, "A sequential stochastic assignment problems," *Management Science*, vol. 18, no. 7, pp. 349–355, Mar. 1972.

[15] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810Ű–837, June 2001.

[16] V. Shestak, J. Smith, H. J. Siegel, and A. A. Maciejewski, "Stochastic robustness metric and its use for static resource allocations," *Journal of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.

[17] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107Ű–121, Nov. 1999.

[18] L. D. Briceno, B. Khemka, H. J. Siegel, A. A. Maciejewski, C. Groer, G. Koenig, G. Okonski, and S. Poole, "Time utility functions for modeling and evaluating resource allocations in a heterogeneous computing systems," in *20th Heterogeneity in Computing Workshop (HCW 2011), in the procedings of the 25th International Parallel and Distributed Processing Symposium (IPDPS 2011)*, May 2011, pp. 7–19.