Automatic and Coordinated Job Recovery for High Performance Computing

Wei Tang,* Zhiling Lan,* Narayan Desai,† Daniel Buettner‡

*Department of Computer Science, Illinois Institute of Technology
Chicago, IL 60616, USA
{wtang6, lan}@iit.edu

†Mathematics and Computer Science Division
‡Argonne Leadership Computing Facility
Argonne National Laboratory, Argonne, IL 60439, USA
† desai@mcs.anl.gov
‡ buettner@alcf.anl.gov

***Abstract*—As the scale of high-performance computing systems continues to grow, the impact of failures on the systems is increasingly critical. Research has been performed on fault prediction and associated precautionary actions. While this approach is valuable, it is not adequate because of the inevitability of failures. Postfailure recovery is equally important; however, most current work relies mainly on checkpoint/restart, not addressing the problem from the system level. We propose AuCoRe, an automatic and coordinated job recovery framework. AuCoRe provides a coordination mechanism for failed-job recovery, taking the execution of regular jobs into account; users specify job recovery policy for their jobs, and an incentive mechanism minimizes gaming. We have implemented AuCoRe in Cobalt, a production resource manager, and evaluated it using real workloads from the Blue Gene/P system at Argonne National Laboratory. Experimental results demonstrate that AuCoRe improves system performance by efficiently managing job recovery.**

I. Introduction

Over the past decades, the insatiable demand for more computational power in science and engineering has driven the development of ever-growing supercomputers. Supercomputers with thousands or hundred thousands proccesors [6] have been deployed to meet the demand of high-performance computing (HPC), high-throughput computing (HTC), and many-task computing (MTC) [24]. The software complexity of such systems has increased as well. Fueled by the ever-growing scale and complexity, these systems often fail in unpredicatable ways. Studies have shown that failure rates of production systems range from 20 to more than 1000 per year. Depending on the root cause of the problem, the system-level mean time to repair (MTTR) can be up to 100 hours or more [19][20][26].

Over the past, intensive research has been conducted on predicting failures and taking precautionary actions before failure occurrence. Since the best fault is the one that does not lead to a failure, extensive study has been done to forecast failures by applying various modeling and data analysis techniques. Despite years of research on failure prediction, failures are inevitable and unexpected failures frequently occur in practice, especially in modern parallel systems with unprecedented sizes and complexities. Meanwhile, numerous methods have been developed for fault tolerance by taking precaution actions ahead of failures. These include checkpointing to mitigate failure impact by periodically saving a snapshot of the system, various proactive methods (e.g., process migration) to avoid anticipated failures, and failure-aware scheduling to reduce possibilities of job failures. While it is important to make every effort to avoid or mitigate failures whenever possible, relying on failure prediction alone is insufficient for fault tolerance due to the inevitability of failures.

Just as failures need to be carefully avoided and tolerated, post-failure recovery (i.e., a procedure taken after failures) is of equal importance and has a profound impact on almost every aspect of high performancel computing, ranging from application performance to system management cost. Nevertheless, little attention has been paid to post-failure recovery in the current research. As a result, failure repair of parallel systems and applications is far from efficient. In the field of high performance computing, failure recovery is mainly relied on checkpoint/restart. When failures occur, users generally have to manually resubmit their failed jobs, thereby resulting in a significantly high MTTR from the user's perspective.

Extensive research has been conducted on resource management and job scheduling. The majority has focused on improving system performance through improved job scheduling or resource allocation policies. Nevertheless, little attention has been paid to provide automated and coordinated failure recovery. Some schedulers like PBS [5] and Moab [4] provide certain simple failed-job restart service for failed jobs but they treat the failed job in the same way and ignore the impact on the regular jobs brought by failed job recovery. In fact, a recovery service that only concerns about failed jobs may negatively impact the performance of regular jobs due to resource limilation. Moreover, failed jobs themselves can be differentiated by priorities. Therefore, the resource manager

978-1-4244-9705-8/10/$26.00

should provide better support for job recovery such that failed jobs will be treated differently according to their importance or priority and the recovery of failed jobs should be coordinated with the scheduling of regular jobs.

In this paper, we present AuCoRe, an automatic and coordinated failure recovery framework for high performance computing. It allows users to specify their preferred job recovery option in the submission script. According to user's specification, AuCoRe will coordinate the recovery of failed jobs with the execution of regular jobs. Such an automatic recovery not only eliminates manual resubmission, but also minimize job delay that may be introduced due to user's response to failure. By coordinating job recovery with job execution, we can fulfill the interests of both failed jobs and regular jobs, thereby maintaining both user satisfaction and good system performance. Specifically, AuCoRe consists of three interrelated parts. First, job scheduler is augmented to allow users to specify options in terms of job recovery during job submission. Second, we have developed an automatic recovery management component to coordinate failed jobs and regular queuing jobs for the use of system resources. Lastly, we have designed an incentive mechanism to encourage users to appropriately set recovery options for their jobs, without gaming the system.

We implement AuCoRe in Cobalt [1], a production batch job scheduler and resource manager that has been used in several supercomputing systems[3]. We evaluate AuCoRe using real workloads from Intrepid, the Blue Gene/P system at Argonne National Laboratory. The experimental results demonstrate that AuCoRe can significantly improve system performance in the presence of failure.

The remainder of the paper is organized as follows. Section II discusses related work. Section III presents the design of AuCoRe. Section IV shows the experimental results, followed by a brief summary in Section V.

II. Related Work

In the past, considerable efforts have been put into the research of fault management in high performance computing, including the area of failure log analysis [19][20][26], failure prediction [27][13][14], checkpoint/restart [10][11][22][28], process migration [9][35], fault-aware job scheduling [31] or rescheduling [18], and adaptive fault management schemes [16]. Most of these methods focus on avoiding or mitigating failure impact by taking precautionary actions before failure occurrence. In contrast to these studies, our study emphasizes post-failure recovery. Hence, it complements the above studies.

The Recovery-Oriented Computing (ROC) Project [21] is closely related to our study. It provides postfailure recovery supports for Internet services by using fine-grained system partition and recursive restart. While being inspired by the ROC project, our research is fundamentally different from the ROC project in two key aspects. First, our target environment is parallel systems used mainly for scientific computing, whose failure patterns are significantly different from those observed in Internet services. Second, while the ROC project emphasizes recovering from human errors through system-level undo and partial system restart, our study focuses on job restart from failures by system orchestration.

In the field of parallel computing, several projects contain failure recovery components. In [25], Rao et al. have proposed a class of hybrid protocols to maintain the failure-free performance of sender-based protocols while approaching the performance of receiver-based protocols during recovery. The FT-MPI project explores five modes for failure recovery of MPI applications [8]. The CiFTS project, a joint research effort led by Argonne, aims to design a fault awareness and notification backplane called FTB to provide a uniform fault-related event handling and notification mechanism among different software components [2]. In [32], the authors present a user-level checkpoint and recovery infrastructure to facilitate automated recovery of user jobs at the Pittsburgh Supercomputing Center. In [34], a job pause mechanism is proposed to restart LAM/MPI applications without job resubmission. Zhang et al. have described a data recovery mechanism for data-intensive computing [37]. In [17], a mechanism is presented to achieve fast job restart from checkpoint image by optimizing the checkpoint/restart procedure. While our study also involves with failed job recovery, it focuses more on a system-wide coordination in the process of failed job recovery.

Since early 1990s, market-based resource management has been discussed in the literature. Spawn [33] is the earliest known work in applying market-based resource management for batch scheduling in distributed systems. The microeconomic parallel scheduler of Stoica et al. [30] also takes an auction-based approach using a first price auction for market-based resource allocation of dedicated CPU time for parallel jobs. Libra[29] is a computational economy-based job scheduling for clusters based on PBS. SHARP[12] deploys market-based methods for resource sharing in an Internet-scale computing infrastructure. Petrou et al. adopt a virtual pricing scheme in their batchactive scheduler[23]. We also adopt incentive management in AuCoRe. Similar to previous work, our scheme also enables users to express their requests and prevent users from gaming the system. The difference is that the credits of most of previous work are used to buy scheduling priority, usage of resources or result data, whereas our credits are used to buy reliability (reflected by the recovery priority), more like insurance.

III. AuCoRe Design

In this section we describe the design of AuCoRe. We first give a system overview, followed by describing recovery options and the incentive mechanism.

A. System Overview

Figure 1 depicts a diagram of AuCoRe. It contains three extensions to the conventional resource management system. First, users are enabled to specify their recovery options in their submission scripts or commands, along with other information such as job wall time and requested nodes.

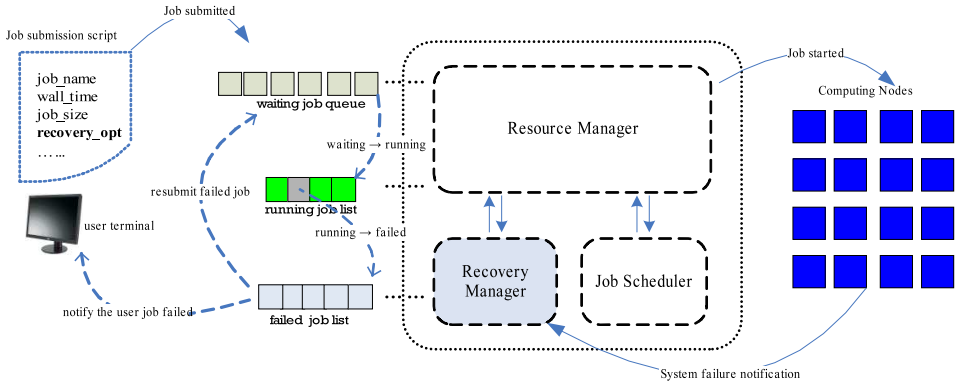

Fig. 1. Diagram of AuCoRe. Users are allowed to specify their job recovery options in job submission scripts or commands. Jobs are maintained in three groups, namely the waiting job queue, the running job list, and the failed job queue. A recovery manager enables automatic and coordinated job recovery and supports an incentive management mechanism.

Second, jobs in the system are separated into three parts: a waiting job queue, a failed job list, and a running job list. When a job is submitted, it is put into the waiting job queue. When a waiting job starts running, it goes to the running job list. When a running job fails, it is moved to the failed job list. In return, when a failed job gets recovered, it is sent back to either the waiting job queue or running job list, depending on recovery actions.

Third, a recovery manager is introduced, in addition to the resource manager and the job scheduler that are commonly included in the traditional resource management systems. The recovery manager performs automatic failed job recovery and coordinates the recovery of failed jobs with regular jobs. It also provides an incentive management mechanism to minimize gaming.

B. Recovery Options

Most batch schedulers implicitly assume that user jobs will run to completion without experiencing any failure. Thus, job scripts include only job attributes used to schedule and run the jobs, such as requested running time and number of computing nodes, without any information from the user's perspective that can be used to recover the jobs in case of failures. that are inevitable in large scale systems. Such an assumption is no longer hold as systems continue to grow in size and complexity.

We argue that users should be allowed to specify the recovery action when their jobs fail, so that important jobs can be treated with higher priority under the resource limitation. To this end, we extend the job scheduler by allowing users to explicitly state job recovery options in their job scripts or job submission commands. For example, if PBS[5] is used, we introduce an additional attribute into the submission scripts like the following:

```
#PBS -N myjob
#PBS -l nodes=1024, walltime=12:00, rec_opt=A
```

In this study, we propose the following recovery options:

- Option A – Notify the user about job failure via e-mails or alarm messages;
- Option B – Automatically resubmit the job to the rear of the waiting queue;
- Option C – Restart the job on the original set of nodes after these nodes get repaired;
- Option D – Automatically insert the job to the middle part of the waiting queue based on its submission time;
- Option E – Automatically resubmit the job to the head part of the waiting queue.

These options reflect different levels of "recovery priorities." By specifying a recovery option, the user can let the system know how critical the job is. With this information, the system understands the user preference when conducting automatic job recovery.

Options A and C involve no apparent competition with regular jobs. Option A indicates lowest recovery priority; a failed job with Option A will be stepped out of the system, waiting for the user to resubmit it manually. These jobs will suffer a period of user response time. A failed job with Option C will be pending on the failed components until they get repaired. These jobs are impacted by the hardware repair time.

Figure 2 illustrates the difference between option B,D, and E. The waiting job queue is logically divided into three parts: the rear, the middle, and the head. Regular jobs are always in the middle. The failed jobs with option E will be automatically put in the head part, thus they have the highest priority to get allocated. The failed jobs with option B will be automatically put in the rear part, and they won't get allocated until all the jobs in the middle are scheduled. When a job with option D fails, it will be automatically inserted in to the middle part based on its original submission time instead of its failed time.

These recovery options are only a proposal in this study. In practice, recovery options should be set or adjusted by the system owners. New options can be added according to system characteristics. For example, for a system that supports preemptive scheduling, a higher-priority option like "Restart the failed job immediately even by preempting other running

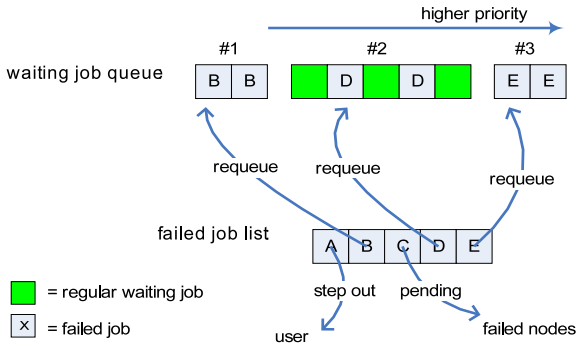

Fig. 2. Treatments for failed job with different recovery options. Option-A jobs are stepped out waiting for manually resubmit; option-C jobs are suspended until computing nodes are recovered; Jobs with option B, D, and E are resubmitted to different part of waiting job queues. Only the higher-priority part of waiting queue is empty can the jobs in lower-priority part be scheduled.

jobs" could be added.

C. Incentive Mechanism

User behavior is hard to model. Some users may ignore specifying any recovery options in their submission scripts. Or worse, some may attempt to gain high priority by always requesting the highest recovery priority. We can address the first issue by providing a default recovery option such as the lowest option.

To address the second issue, we propose *a novel virtual credit incentive mechanism*. It is very similar to an insurance system. Users are required to pay a premium for recovery service, and the price corresponds to recovery priority. They are charged regardless of the occurrence of system failures.

1) Pricing: Each recovery option is associated with a recovery price, based on the quality of service. In AuCoRe, the cost for a job with recovery option i is defined as

$$C = \alpha_i \times T \times N, \tag{1}$$

where α_i is the unit price of Option i, T is the job's running time (in hour), N is the number of the job's computing nodes. α_i is set by system owners, based on the principle that a higher unit price corresponds to a higher recovery priority.

Clearly, the recovery priority of options A, B, D, and E is in an increasing order. But the relative priority of Option C is determined by the system MTTR (mean time to repair). It could be lower than option B, higher than option E, or in between. The system owner can set the unit prices according to the real system parameters.

2) User Recovery Account: Most high performance computing centers adopt a virtual currency system. Accrual of credits at a constant rate is a common mechanism [30]. This approach, however, can be less effective when user jobs are not evenly distributed; users may run out of money easily during the period that they really need it, while accumulating lots of money when they do not need it.

To address the problem, AuCoRe assigns credits to each user based on his/her job submission. Each time a user submits a job, he is assigned a certain amount of credits:

$$S = \beta \times T \times N, \tag{2}$$

where β is a parameter predefined by system owners. For instance, we may set it as the median unit price (P_m) of all the options. For example, assuming the prices for each of five recovery options listed in Section III-B are 0 to 4, then P_m is 3. This number means that each time a user submits a job, he is allocated with the credits affording one recovery Option C. But the user needs not to specify the job with option C every time. He can choose a cheaper one so that he will deposit some credits in the saving account for later uses, or he can make an overdraft to pay a more expensive one. There is a maximum deposit limit M and a maximum overdraft limit M' for each account.

This scheme is resilient to user abuse. If a user aggressively uses high-price recovery options, his overdraft amount will soon reach the limit M', stopping him from doing this. If a user keeps using low-price options, he will get the credits accumulated, but no more than the deposit limit M. As a result, no one can get rich by intentionally saving credits.

3) Charging: Charge is made at job completion when the actual job runtime is available. A user is charged for the job with option i as follows:

$$B = (\alpha_i - \beta) \times T \times N. \tag{3}$$

Note that B may be negative, which means the user accumulates B credits at this run. By using actual runtimes instead of user estimates, we prevent users from gaining credits by repeatedly submitting faked jobs with low options and aborting them right after they start. We ignore the case that users run faked low-option jobs to accumulate credits, because this will waste their compute allocations (or node-hours) that are limited per project (this is the allocation policy adopted by most DOE and NSF supercomputing centers).

With this incentive scheme, users are encouraged to specify proper recovery options for their jobs based on the job importance, without a chance to game with the system. From a global system view, the currency scheme promotes diversity in terms of recovery options, which is beneficial for recovery coordination as we will show in our experiments.

IV. Experiments

We have implemented AuCoRe in Cobalt [1][31], a production resource management system mainly used on Blue Gene systems, by adding a new recovery management component and extending the job submission interface (See Figure 3). In this section, we evaluate the AuCoRe implementation using trace-based simulations. The event-driven job scheduling simulator included in the Cobalt distribution has been modified for our experiments. We first describe our experimental setup and evaluation metrics, and then present the results.

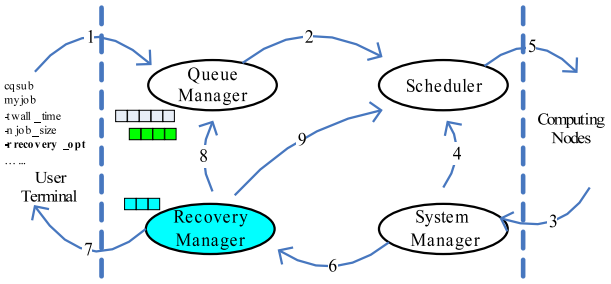

Fig. 3. AuCoRe Implementation with Cobalt. A newly introduced recovery manager component works together with three legacy Cobalt components – queue manager, scheduler, and system manager. Under normal condition, the queue manager gets jobs from user (1), and provides job information to the scheduler (2). The system manager obtains resource information (3) and provides it to the scheduler (4). Based on the job and resource information, the scheduler periodically makes scheduling decision, starting certain jobs on some computing nodes (5). When failure occurs, the system manager notifies the recovery manager (6). The recovery manager then takes actions for the failed jobs according to their recovery options, either by notifying the users (7), inserting the jobs to the waiting queue (8), or requesting the scheduler to restart them immediately (9).

A. Experimental Configuration

1) Simulation Cases: We use trace-based simulations to evaluate AuCoRe-enhanced job scheduling as against the regular job scheduling without AuCoRe. For convenience, we call a simulation with certain configuration a "case". Two category of cases are examined: one is that without automatic recovery, the other is with automatic recovery. By doing so, we can evaluate how much improvement the automatic recovery can bring. In the cases with AuCoRe, we also distinguish multi-option cases and single-option cases, in order to quantify the difference of ways to treat failed jobs.

Table I summarizes the cases conducted in our experiments. As shown in Table I, two cases are without AuCoRe, namely FF and MR. FF means "failure free"; in this case all jobs run to completion without experiencing failure, thus it indicates an ideal case. MR means "manual resubmission." To simulate this case, we assume the failed job will be "manually" resubmitted when the estimated job completion time (wall time) expires. The reason for using this time is that we assume that a user typically check his job results around the expiration of his job walltime, and if he finds the job failed, he will resubmit it immediately. MR represents a common scenario occurring in practice now.

For the cases with AuCoRe, we separate the cases into two groups: multi-option representing the cases where multiple recovery options are used and single-option representing the cases where a single option is used for all the jobs. Two multi-option cases, namely "Even" and "Normal", are simulated. "Even" means all the options are used and of equal proportion. "Normal" means the jobs' recovery options are like normal distribution (the proportion of each option is 1:2:4:2:1). The single-option cases include four scenarios such that all jobs use the same option. We do not list All-A because it is equivalent to MR.

We assume all jobs adopt application level checkpoint/restart, which is supported by Blue Gene/P systems [3]. We use the WFP/Backfilling [31] scheduling policy used in the production Blue Gene/P system at Argonne.

TABLE I
DENOTATIONS OF SIMULATION CASES

Cases	Denote	Description
W/O AuCoRe	FF	failure-free
	MR	failure-present, manual resubmit
W/ AuCoRe (multi-opt)	Even	Option proportion is 1:1:1:1:1
	Normal	Option proportion is 1:2:4:2:1
W/ AuCoRe (single-opt)	All-B	all with option B
	All-C	all with option C
	All-D	all with option D
	All-E	all with option E

2) Job Trace: We collected a job trace from the 40-rack Blue Gene/P system at Argonne National Laboratory. The machine is known as Intrepid. It contains 40,960 quad-core nodes, which is ranked the 9th in the latest TOP500 list [6] released in June 2010. The trace contains around 11000 jobs within one and a half months. The average job running time is around 100 minutes. The job size varies from 512 nodes to 32,768 nodes.

3) Failure Trace: Since Intrepid is a relatively new system, we do not have its steady-state failure rate. Recent studies have shown that in most production systems, failure typically follows Weibull distributions. In this study, we generated a failure trace which follows Weibull distribution, with a node-level MTBF varying from 150,000 to 1,000,000 hours. For a 40,960-node system, it means the system-wide MTBF ranges between 4 to 24 hours. This range covers the numbers presented in recent studies of various production HPC systems.

B. Evaluation Metrics

We evaluated the impact of AuCoRe on failed jobs as well as on overall system performance.

To measure its impact on failed jobs, we use average failure slowdown (FSD), which is defined as the ratio of time delay caused by failure to failure-free job execution time. We measure the average slowdown over all the failed jobs. Instead of the number of total jobs, we use the number of failed jobs as denominator, in order to indicate impact on the failed jobs.

To measure its overall system performance we use average response time (RESP). A jobs response time is the time from jobs submission to its completion. In the failure free context, response time includes queuing time and execution time. In the failure present context, all the delay caused by failure interruption is also counted in. We use average response time to measure the overall system performance. We do not list slowdown, the ratio of response time to the running time, because its trends are similar to RESP.

C. Results

1) Baseline Results: Our baseline configuration is summarized in Table II. These parameters and their corresponding

ranges are chosen according to the results reported in [36][3] and our experiences [16][7].

TABLE II
BASELINE CONFIGURATION

System-wide MTBF	10 hours
Average job arrival interval	5 Minutes
Node-level MTTR	60 minutes
Job checkpoint overhead	5 minutes
Job checkpoint inteval	50 minutes

Figure 4 presents the baseline results. The x-axis represents simulation cases described in Table I. As shown in Figure 4(a), system performance, in case of failure, is degraded compared with FF. MR has FSD value 1.71, meaning that failed jobs in average will use 71% more time to complete in case of failure.When using automatic job resubmission, the FSD value decreases to the range 1.06 through 0.47, i.e., improved by 38% to 73%, which is very significant. Specifically, from All-B to All-E, the FSD decreases monotonously. All-E is the best and All-B is the worst; Even and Normal are in between. In a word, a higher recovery option achieves higher recovery performance when using single options; using multiple options achieves medium recovery performance regarding FSD.

As shown in Figure 4(b), RESPs for FF and MR are 169 and 235 minutes respectively, meaning that system performance is degraded by 28% because of failures. The RESP for AuCoRe cases range from 185 (Normal) to 216 (All-D), i.e., improved by 8% to 21% compared with MR. RESP improvements for different cases are similar, except that All-D and All-E are slightly worse than others. Overall, AuCoRe can mitigates impact of failures on system performance, no matter how to assign recovery options.

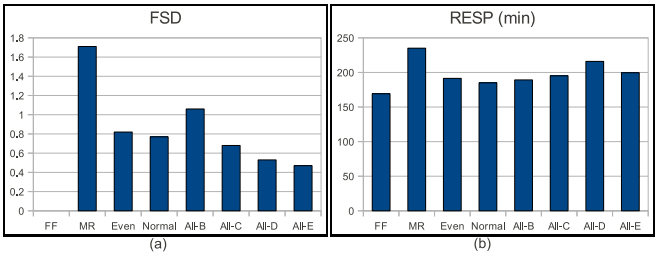

Fig. 4. Baseline Simulation Results.

From Figure 4 we can see All-D and All-E have lowest FSDs. So why not just use All-D or All-E instead of multioption cases? One reason can be seen in Figure 4 is that All-D and All-E have slightly worse RESP but it is not compelling enough. Figure 5 illustrates the advantage of multi-option over single-option cases.

First, we examined how differently recovery options influence the recovery performance in the multi-option cases. To quantitatively evaluate it, We divide all the jobs into five groups by their recovery options (jobs with the same recovery option are grouped together). And we examine the FSD values for each job group. In Figure 5, the x-axis represents the job groups, denoted by their recovery options. From Even and Normal bars in Figure 5(a), we can get the FSD values for each job group. For example, jobs with recovery option A have FSD value 1.98 and 2.19, for Even and Normal respectively (leftmost two bars); jobs with recovery option E have FSD value 0.36 and 0.30 respectively. If we examine one particular case, either Even or Normal, the FSD value decreases as the recovery option gets higher. That is, in a multi-option case simulation, jobs with higher recovery option achieve better recovery performance in case of failure.

Second, we compared the FSD value of each job group with the FSD value of jobs from corresponded single-option cases. The All-X bars in Figure 5(a) are added for the comparison. From the figure, we can compare jobs of option X in multi-option case with jobs in single-option case All-X. For example, FSD for jobs in All-A is smaller than jobs of option A in Even and Normal. Similarly, jobs of option B in Even and Normal also have worse FSD than jobs in All-B, but jobs of option C, D, E in multi-option cases have better FSD than jobs in corresponded All-C, All-D, and All-E, respectively.

Figure 5(b) presents a more straightforward view about the relative gains of jobs in multi-option cases (Even and Normal) compared with corresponding single-option case. As shown in the figure, for option A and B, the gain is negative; for option C, D, and E, the gain is positive, with the gain of option E up to 36%. That is, high-option jobs in multi-option cases get good recovery performance, even better than jobs in the corresponded single-option cases (All-D, All-E). This is because that in multi-option cases, high-option jobs recovery priority is more safely assured by sacrificing part of lowoption jobs. Imagine that in All-E or All-D, though all the jobs are treated with same high priority, their priority may not be guaranteed because of resource limitation. Therefore, assigning jobs with different recovery priority can benefit part of jobs which users think really important. In this context, treating failed jobs differently is better than treating all the jobs the same. Fortunately, the incentive mechanism prevents all the jobs from being assigned with the same highest recovery priority thereby maintaining diversity of recovery options.

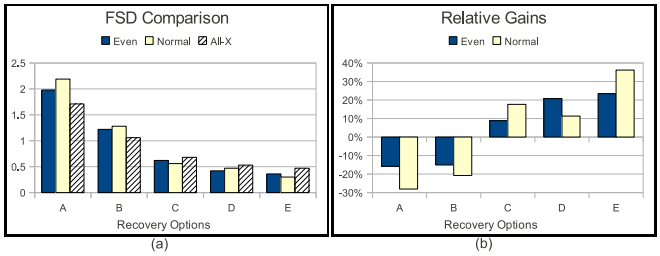

Fig. 5. Comparing multi-option cases with single-option ones. The X-axis represents the job groups categorized by their recovery options.

2) Impact of System MTTR: Node repair time is essential to the whole system performance in the presence of failure. In

this study, Node MTTR will directly influence the recovery performance of jobs with Options C, and will impact the system performance as a whole. By tuning the node MTTR arguments in the simulations, we evaluated the impact of system MTTR. Note that except MTTR, other parameters are not changed in Table II. As shown in Figure 6, MTTRs are tuned from 240 minutes to 15 minutes. For better illustration, we separate single-option cases with multi-option cases; MR is compared with both categories.

Figure 6(a) and 6(b) present the FSD trends. As shown in the figures, MR trend is less related to MTTR. This may because in MR case, the job delay is more attributed to user response before manual resubmission. In other cases, FSD decreases as MTTR decreases. The overall trend shows that a small MTTR corresponds to a small FSD. We also observe that the FSD of All-C is most sensitive to the MTTR change: at 240 minutes, the FSD of All-C is 2.76, very close to MR; at 120 minutes, it decreases significantly to 1.21 but is still second worst; at 60 minutes, it is between All-B and All-D; at 30 minutes, it is second best, only larger than All-E; and at 15 minutes, the FSD of All-C is 0.12, best among all cases. This trend indicates that when assigning unit price for recovery options, system administrator should consider the MTTR of the system. For example, in a small MTTR system, unit price for option C should be high. The relative unit price of option C with other options can be obtained from experience or simulations.

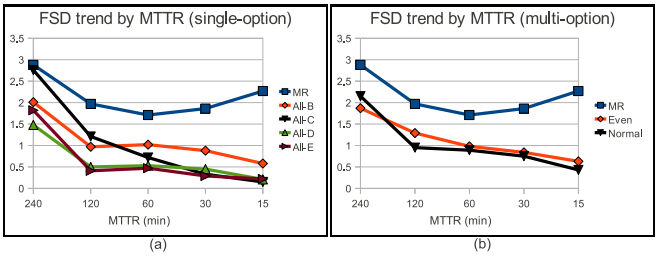

Fig. 6. Performance under different MTTR.

3) Impact of System MTBF: To examine the performances sensitivity to failure rate, we simulate each AuCoRe under various node-level MTBFs. Here we only compare the multi-option cases with MR, since the trends of single-option cases are also similar.

As shown in Figure 7(a), FSD increases as the MTBF decreases. Specifically, under low failure rate, the change is not as significant as under high failure rate. Because FSD is the average value among the failed jobs, so the number of failed jobs will not impact much on FSD. So the increase of FSD may be caused by the fact that a single job may be interrupted by failures more than one time. This is more likely to happen under high failure rates.

As shown in Figure 7(b), RESP increases as the MTBF decreases, with the exception that under low failure rates, MR even has lower RESP than Even and Normal. One explanation is that sometimes preemptive starting jobs can benefit average response time; jobs interrupted by failure can be considered as being preempted. Overall, AuCoRe can mitigate the failure impact on system performance under various failure rates.

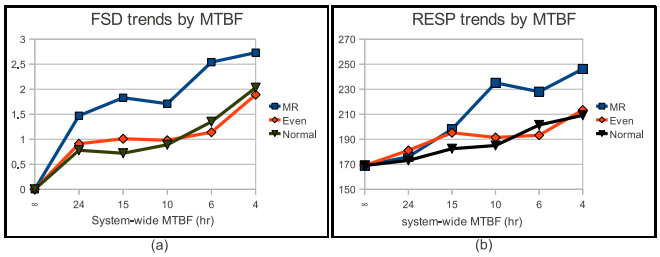

Fig. 7. Performance under different system MTBF.

4) Impact of Job Arrival Rate: To make our results more representative, we tune the job arrival intervals of our original job trace, and repeat simulations using different workloads. The MTBF and MTTR use the baseline configuration. Again, we only exam multi-option cases and MR here. We use the metric average job arrival interval to represent job arrival rate. Less job arrival interval represents higher job arrival rate.

As shown in Figure 8, as the mean job arrival interval gets higher, RESP decreases and FSD fluctuates. We can also see that using AuCoRe mitigates the impact of failure in all cases, especially on the metric FSD.

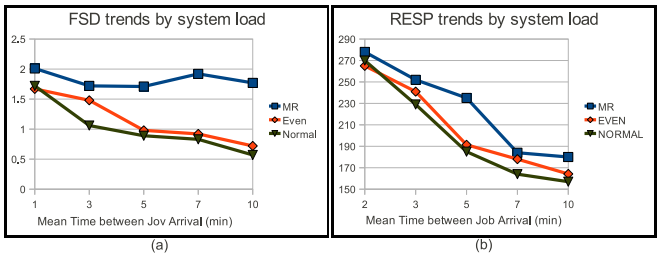

Fig. 8. Performance under different job arrival rates.

5) Results Summary: In summary, we have made the following observations based on the above experiments:

- AucoRe can significantly improve performance of failed jobs and the overall system performance, compared with the common approach that relies on manual job resubmission. Specifically, under our baseline setting, the FSD and RESP can be improved by up to 73% and 21%, respectively, compared with MR.
- In the multi-option cases, higher-priority recovery options result in more performance gains than lower-priority options, especially on FSD. And, having recovery option diversity can benefit part of jobs that are really thought important. Specifically, FSD of jobs with option E in multi-option case can achieve up to 36% enhancement than FSD of jobs in single-option case All-E. Therefore, though single-option case is easy to deploy, assigning jobs with diverse options has its merit and is recommended.
- The recovery performance is sensitive to MTTR. Therefore, when setting the relative unit price of option C,

MTTR should be considered.
- AuCoRe is effective under different system failure rates and job arrival rates.

V. Summary

We have presented AuCoRe, an automatic and coordinated job recovery framework. With this framework, users can specify recovery options for their jobs at job submission. When the jobs fail, the system performs automatic job recovery according to the user-specified recovery options, taking into account of the global system performance via recovery coordination. AuCoRe also supports an incentive mechanism that encourages users to properly specify recovery options. We have implemented AuCoRe as an extension to Cobalt, a production resource manager mainly used for Blue Gene systems. Our trace-based simulations using a real job trace from Blue Gene/P system at Argonne have demonstrated that AuCoRe can significantly improve the failed-job recovery performance measured by the average failure slowdown. Meanwhile, the system level performance, measured by average bounded slowdwon is also improved. In sum, AuCoRe is complementary to the conventional research on pre-failure prediction and tolerance. While prefailure prediction and tolerance can be used to avoid or mitigate failure by taking actions before failure occurrence, AuCoRe can effiently resume user jobs in the presence of failure.

This study is our first step toward recovery-aware parallel computing. Motivated by the promising results as well as some limitations of the current work, we have several next steps. First, we plan to design the recovery management scheme not only using user specified recovery option, but also considering the system status to coordinate job recovery with regular job scheduling. Second, we plan to deploy and test AuCoRe-enhanced cobalt system at the production Intrepid system at Argonne. Last, we plan to integrate AuCoRe with fast failure diagnosis which is an on-going project in our group [15].

Acknowledgments

This work is supported in part by US National Science Foundation grants CNS-0834514, CNS-0720549, and CCF-0702737. The work at Argonne National Laboratory is supported by DOE Contract DE-AC02-06CH11357.

References

[1] Cobalt project. *http://trac.mcs.anl.gov/projects/cobalt*
[2] Coordinated Infrastructure for Fault Tolerant Systems (CIFTS). *http://www.mcs.anl.gov/research/cifts/index.php*
[3] Blue Gene Team, "Overview of the IBM Blue Gene/P project," *IBM Journal of Research and Development*, 2008.
[4] Moab resource manager. *http://adaptivecomputing.com*
[5] Portable Batch System(PBS). *http://www.openpbs.org*
[6] TOP500 Supercomputing. *http://www.top500.org*.
[7] N. Desai, R. Bradshaw, C. Lueninghoener, A. Cherry, S. Coghlan, and W. Scullin, "Petascale System Management Experiences", *Usenix Large Installation System Administration Conference(LISA'08)*, 2008.
[8] D. Dewolfs, J. Broeckhove, V. Sunderam, and G. Fagg, "FT-MPI, Fault-Tolerant Metacomputing and Generic Name Services: A Case Study", *Lecture Notes in Computer Science*, ICL-UT-06-14, Vol.4192, 2006.
[9] C. Du, X.-H. Sun and M. Wu, "Dynamic Scheduling with Process Migration",*Proc. of IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, 2007.
[10] J. Duell, P. Hargrove, and E. Roman, "Requirements for Linux Checkpoint/Restart," *Technical Report LBNL-49659, Berkeley Lab*, 2002.
[11] E. Elnozahy and J. Plank, "Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery", *IEEE Transaction on Dependable and Secure Computing*, 2004.
[12] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat, "SHARP: An Architecture for Secure Resource Peering," *ACM SOSP'03*, 2003.
[13] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A Meta-Learning Failure Predictor for Blue Gene/L Systems," *Proc. of International Conference on Parallel Processing (ICPP'07)*, 2007.
[14] J. Gu, Z. Zheng, Z. Lan, J. White, E. Hocks, and B. Park, "Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A Case Study," *Proc. of International Conference on Parallel Processing(ICPP'08)*, 2008.
[15] Z. Lan, Z. Zheng and Y. Li, "Toward Automated Anomaly Identification in Large-Scale Systems," *IEEE Transaction on Parallel and Distributed Systems*, 2009.
[16] Z. Lan and Y. Li, "Adaptive Fault Management of Parallel Applications for High Performance Computing," *IEEE Transaction on Computers*, vol. 57, no. 12, pp. 1647-1660, 2008.
[17] Y. Li and Z. Lan, "A Fast Restart Mechanism for Checkpoint/recovery protocols in networked environments," *Proc. of DSN'08*, 2008.
[18] Y. Li, Z. Lan, P. Gujrati, and X. Sun, "Fault-Aware Runtime Strategies for High Performance Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 4, pp. 460-473, 2009.
[19] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo,"Blue Gene/L Failure Analysis and Models," *Proc. of DSN'06*, 2006.
[20] A. Oliner and J. Stearly, "What Supercomputers Say: A Study of Five System Logs," *Proc. of DSN'07*, 2007.
[21] D. Patterson et al., "Recovery-Oriented Computing (ROC): Motivation, definition, techiques, and case studies," *UC Berkeley Computer Science Technical Report UCB//CSD-02-1175*, 2002.
[22] F. Petrini, K. Davis and J. Sancho, "System-Level Fault-Tolerance in Large-Scale Parallel Machines with Buffered Coscheduling," *Proc. of International Parallel and Distributed Processing Sysmposium (IPDPS'04)*, 2004.
[23] D. Petrou, G. Gibson, and G. Ganger, "Scheduling speculative tasks in a compute farm," *Proc. of ACM/IEEE SC'05*, 2005.
[24] Ioan Raicu, Ian Foster, Yong Zhao. "Many-Task Computing for Grids and Supercomputers," *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08)*, 2008.
[25] S. Rao, L. Alvisi and H. Vin, "The Cost of Recovery in Message Logging Protocols," *IEEE Transaction on Knowlegde and Data Engineering*, vol. 12(2), 2000.
[26] B. Schroeder and G. A. Gibson, "A Large-scale Study of Failures in High Performance Computing Systems," *Proc. of DSN'06*, 2006.
[27] R. Sahoo, A. Oliner, et al., "Critical Event Prediction for Proactive Management in Large-scale Computer Clusters, " *Proc. of International Conference on Knowledge Discovery and Data Mining*, 2003.
[28] M. Schulz, G. Bronevetsky, R. Fernandes, D. Marques, K. Pingali, and P. Stodghill, "Implementation and Evaluation of a Scalable Application Level Checkpoint-Recovery Scheme for MPI Programs," *Proc. of ACM/IEEE SC'04*, 2004.
[29] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya, "Libra: a computational economy-based job scheduling system for clusters," *Software - Practice and Experience*, vol 34: pp 573-590, 2004.
[30] I. Stoica, H. Abdel-Wahb, and A. Pothen, "A Microeconomic Scheduler for Parallel Computers," *Springer-Verlag Lecture Notes in Computer Science*, Vol.949, pages 200-218, 1995.
[31] W. Tang, Z. Lan, D. Desai, and D. Buettner, "Fault-aware, utility-based job scheduling on Blue Gene/P system," *Proc. of IEEE Cluster'09*, 2009.
[32] N. Stone, J. Kochmar, R. Reddy, J. Scott, J. Sommerfield, and C. Vizino, "A Checkpoint and Recovery System for the Pittsburgh Supercomputing Center Terascale Computing System," *PSC technical report CMU-PSC-TR-2001-0002*, 2001.
[33] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and S. Stornetta, "Spawn: A distributed computational economy," *IEEE Transactions on Software Engineering*, 18(2):103-177, 1992.
[34] C. Wang, F. Mueller, C. Englemann, and S. Scott, "A Job Pause Service under LAM/MPI+BCVR for Transparent Fault Tolerance," *Proc. of IEEE IPDPS'07*, 2007.

[35] C. Wang, F. Mueller, C. Engelmann, and S. Scott, "Proactive Process-Level Live Migration in HPC Environment," *Proc. of ACM/IEEE Supercomputing (SC'08)*, 2008.
[36] Y. Zhang, M. Squillante, A. Sivasubramaniam and R. Sahoo, "Performance Implications of Failures in Large-Scale Cluster Scheduling," *Proc. of Workshop on Job Scheduling Strategies for Parallel Processing*, 2004.
[37] Z. Zhang, C. Wang, S. Vazhkudai, X. Ma, G. Pike, J. Cobb, and F. Mueller, "Optimizing Center Performance through Coordinated Data Staging, Scheduling and Recovery," *Proc. of ACM/IEEE Supercomputing (SC'07)*, 2007.
[38] Z. Zheng, Z. Lan, B.-H. Park, and A. Geist, "System log pre-processing to improve failure prediction," in *Proc. of DSN'09*, 2009.