Checkpointing vs. Migration for Post-Petascale Supercomputers

Franck Cappello*, Henri Casanova†, Yves Robert‡

* INRIA-Illinois Joint Laboratory for Petascale Computing
Urbana-Champain, Illinois, U.S.A.
fci@lri.fr
† Information and Computer Sciences Dept.
University of Hawai'i at Manoa, Honolulu, U.S.A.
henric@hawaii.edu
‡ Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon, France
Yves.Robert@ens-lyon.fr

***Abstract*—An alternative to classical fault-tolerant approaches for large-scale clusters is failure avoidance, by which the occurrence of a fault is predicted and a preventive measure is taken. We develop analytical performance models for two types of preventive measures: preventive checkpointing and preventive migration. We also develop an analytical model of the performance of a standard periodic checkpoint fault-tolerant approach. We instantiate these models for platform scenarios representative of current and future technology trends. We find that preventive migration is the better approach in the short term by orders of magnitude. However, in the longer term, both approaches have comparable merit with a marginal advantage for preventive checkpointing. We also find that standard non-prediction-based fault tolerance achieves poor scaling when compared to prediction-based failure avoidance, thereby demonstrating the importance of failure prediction capabilities. Finally, our results show that achieving good utilization in truly large-scale machines (e.g., 2^{20} nodes) for parallel workloads will require more than the failure avoidance techniques evaluated in this work.**

***Index Terms*—failure prediction; checkpointing; migration; parallel jobs;**

I. Introduction

Resilience is one of the key challenges for post-petascale high-performance computing (HPC) systems [1], [2]. The two main approaches are *fault tolerance* and *failure avoidance*. Fault tolerance typically combines redundancy and rollback recovery. Low-overhead redundancy mechanisms detect and correct local faults (e.g., memory error, arithmetic error). When such faults cannot be corrected, rollback recovery is used to resume execution from a previously saved fault-free execution state. Rollback recovery leads to non-negligible overhead during fault-free execution, and to the loss of recent execution progresses when a fault occurs. Checkpoint files can be saved on a remote file system, so that a task can be resumed from a saved checkpoint on any node in the system. On current large-scale clusters (i.e., those in the Top500 [3]), a checkpoint on a disk that is under the control of an I/O node is limited by bandwidth to that I/O node and takes typically on the order of 25 minutes. Recent reports [4], [5] show that this approach, when it relies on coordinated checkpointing, will simply not scale sufficiently for future exascale machines. Consequently, in this work we only consider local checkpoints. This means that a task can resume execution from a checkpoint file stored on the node on which this file was created.

Failure avoidance consists in avoiding the effects of faults. These fault effects can be flagrant (e.g., process execution failure, kernel panic) or more subtle (e.g., uncorrectable memory error, software bug) and may propagate either vertically in the software stack or/and horizontally onto other nodes. In most cases, they eventually lead to incorrect results if they are not confined. Failure avoidance is possible only if faults can be predicted, in which case preventive actions can be taken before their effects happen or propagate. Prediction is generally based on detected non-fatal "precursor events" [6]. Fault predictors do not need to predict root causes of failures but they need to detect at least their early effects.

Failure avoidance for parallel execution in HPC systems usually uses *preventive migration* [7], [8]: the state

of a task is migrated to another node when a fault is suspected to happen on the current node in the near future. Preventive migration is appealing because it lets the execution continue (as opposed to fault tolerance via rollback recovery), but it suffers from four limitations:

1) it must be combined with some form of rollback recovery to handle false negatives (faults that the predictor fails to predict);
2) migration involves moving large amounts of memory content between two nodes, and RAM size per node is expected to increase in the next decade (even if the memory per core is likely to decrease);
3) execution is slowed down or paused during migration;
4) migration requires that a subset of the cluster's node be reserved as spares.

An alternative to migration that addresses these limitations is *preventive checkpointing*: fault prediction is used to trigger a (local) checkpoint before the fault effects propagate to the application tasks on the same node. If the fault is hardware, the potential victim tasks are restarted on other nodes. If the fault is software (but not coming from the application), preventive software rejuvenation [9] is used before restarting the application task. Software rejuvenation is often simply achieved via rebooting [10]. The state of a node after a rejuvenation is assumed to be equivalent to that of a node to which a task would migrate if preventive migration were used, i.e., free of software error. Preventive checkpoint-restart has its own limitations:

1) it triggers checkpoints-restart even for false positives (predicted faults that do not occur);
2) execution is interrupted during reboot;
3) spare nodes are needed to cope with hardware faults.

In this paper we compare preventive migration and preventive checkpointing in the context of HPC applications. We focus the comparison on software faults because hardware failures will require tasks to be moved to spare nodes for both approaches. Note that software faults account for between 40% and 75% of faults in HPC systems [5], [11]. Section II defines the problem, establishes notations, and introduces a performance metric. Our main contributions are analytical performance models for both approaches (Sections III and IV). Section V presents numerical results for current and likely future platform scenarios. Section VI provides a brief comparison to non-prediction-based fault tolerance. Section VII revisits results in Section VI with the additional assumption that parallel job sizes are limited to a fraction of the platform. Finally, Section VIII summarizes our main findings and gives future research directions.

II. Problem Definition

We consider a cluster that consists of N identical nodes and that is used to run sequential or parallel jobs. For simplicity we assume that, in a fault-free scenario, no node ever stays idle, meaning that job scheduling is perfect. It would, however, be possible to account for sub-optimal cluster utilization in our performance models. Each cluster node is subject to software failures with mean time to failure (MTTF) μ, in seconds. We leave the particular probability distribution unspecified. We assume perfect fault predictors so that preventive action can be taken "just before" the failure would occur. This assumption is made for both failure avoidance approaches, which are thus both evaluated in a best-case scenario.

A task of a job running on a node can be checkpointed locally on that node in C seconds, and the checkpoint can be loaded back into memory in R seconds. The time to migrate a task to another node is M seconds. These values depend on the size of the task's memory footprint. The general consensus is that the memory footprint of tasks in HPC jobs is much lower than that available on a cluster node [12], [13], [14], [15], [16]. Unfortunately, no explicit analytical model is available in the literature. It such a model were available, it would be straightforward to integrate it with our performance models. Instead, we assume that C, R, and M do not depend on the particular task and that, pessimistically, each task uses all the available memory on the node on which it executes. Finally, the downtime of a node, i.e., the time to reboot it for software rejuvenation, is D seconds. If $M \geq C+D+R$, then preventive migration is never advantageous when compared to preventive checkpointing using a node as its own space. In practical current and foreseeable scenarios, however, $M < C + D + R$.

Our performance metric is the *expected throughput*, ρ, defined as the expected value of the number of nodes that are performing useful work (i.e., not engaged in checkpointing, checkpoint recovery, task migration, or rebooting) at a given time. The optimal value, achievable in a fault-free environment, is N. Our objective is to obtain analytical expressions for the expected throughput for both migration- and checkpoint-based failure avoidance approaches.

III. Sequential Workloads

In this section we assume that each job uses only one node. The probability, u_c, that a node at a given time is not rebooting, checkpointing, or performing a checkpoint

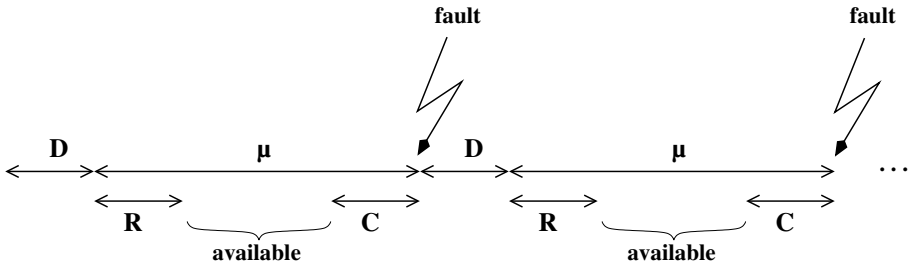

Figure 1. Node execution time-line with preventive checkpointing.

recovery is

$$u_c = \max\left(0, \frac{\mu - R - C}{\mu + D}\right) .$$

This equation is explained in Figure 1. A node goes through a sequence of time intervals: intervals of duration D during which the node is being rebooted, and intervals of (average) duration μ during which the node is available for computation before a fault occurs. At the beginning of the interval of duration μ a checkpoint recovery, which takes take R seconds, must occur. At the end of the interval of duration μ, a checkpoint must be taken, which takes C seconds. Recall that we optimistically assume that the checkpoint is taken exactly before the fault occurs, as depicted in the figure. Each of the N nodes is available for (useful) computation on average $\max(0, \mu - R - C)$ seconds each $\mu + D$ seconds. The maximum is to account for the case in which $\mu < R + C$. In this case the node cannot perform any useful computation. Hence, the throughput of preventive checkpointing, ρ_c, as

$$\rho_c = u_c \times N = \max\left(0, \frac{\mu - R - C}{\mu + D}\right) \times N . \quad (1)$$

With preemptive migration a task must be migrated to a spare node that is immediately available. We denote by n the number of nodes that are reserved as spares out of the N nodes available in the cluster. The probability, u_m, that at a given time a node is not migrating a task to a spare node and not rebooting is

$$u_m = \max\left(0, \frac{\mu - M}{\mu + D}\right) .$$

This equation is similar to the equation for u_c and is explained in Figure 2. As before, a node goes through a sequence of time intervals: intervals of duration D during which the node is being rebooted, and intervals of (average) duration μ during which the node is available for computation before a fault occurs. At the end of the interval of duration μ, the task executing on the node must be migrated to a spare node, which takes M seconds. The node then reboots and becomes a spare node. Therefore, each of the $N - m$ nodes is available for (useful) computation on average $\max(0, \mu - M)$ seconds each $\mu + D$ seconds. The maximum is to account for the case $\mu < M$. In this case the node cannot perform any useful computation. Hence the throughput of preventive migration, ρ_m, as

$$\rho_m = u_m \times (N - n) = \max\left(0, \frac{\mu - M}{\mu + D}\right) \times (N - n) . \quad (2)$$

With preventive migration, it is possible that a task needs to be migrated but that no spare node is currently available. In this case, the job to which this task belongs fails and must be restarted from scratch. The probability that there is no shortage of spares is

$$success(n) = \sum_{k=0}^{n} \binom{N}{k} u_m^{N-k} (1 - u_m)^k . \quad (3)$$

Each term in this sum corresponds to all possible subsets of size k of all available nodes, up to n nodes, so that those k nodes are engaged in preventive migration or reboot, which occurs at a given time with probability $(1 - u_m)$ for each node. The goal is to choose a good fraction of nodes as spares ($n = \alpha(\varepsilon)N$), so that $success(n) > 1 - \varepsilon$. While $\alpha(\varepsilon)$ cannot be computed analytically based on the previous equation, it can be computed numerically efficiently using a binary search.

IV. Parallel Workloads

The work in [17] proposes and validates models of parallel jobs. In particular, the number of nodes used by a job is shown to follow a two-stage log-uniform distribution biased toward powers of two, with roughly a quarter of the jobs being sequential. Based on this observation, we use the following model. We assume that $N = 2^Z$. The probability that a job is sequential is $\alpha_0 = 0.25$. Otherwise, a job is parallel and the probability that it uses 2^j nodes is independent of j and equal to $\alpha_j = (1 - \alpha_0)/Z$.

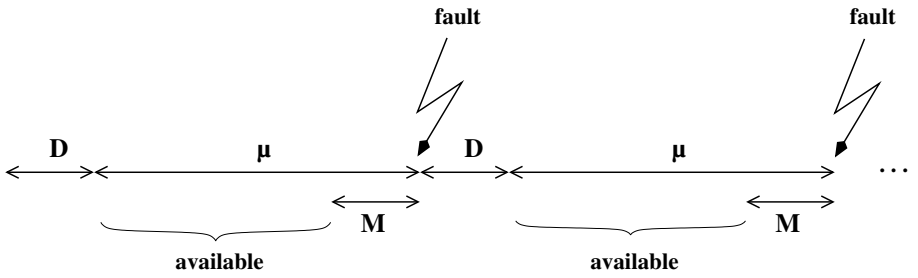

Figure 2. Node execution time-line with preventive migration.

Since we assume a steady-state full utilization of the cluster, we can compute the expected number of jobs that use 2^j nodes, which we denote by β_j, for $0 \leq j \leq Z$. Let K denotes the total number of jobs, then $\beta_j = \alpha_j K$. Due to our assumption that the cluster is fully utilized, we can write that the total number of available nodes, N, is equal to the total number of nodes used by all running jobs:

$$N = \sum_{j=0}^{Z} 2^j \alpha_j K \quad (4)$$

$$= K\left(\alpha_0 + \frac{1-\alpha_0}{Z}\sum_{j=1}^{Z} 2^j\right) \quad (5)$$

$$= K\left(\alpha_0 + \frac{1-\alpha_0}{Z}(2N-2)\right) . \quad (6)$$

The last equation gives K as a function of N, Z, and α_0, which are all fixed quantities. We can thus compute β_j for $0 \leq j \leq Z$.

We can now compute the expected throughput for preventive checkpointing. If a job uses 2^j nodes, we use μ_j to denote the expected interval of time between two faults. We consider that each parallel job is tightly-coupled, meaning that it can make progress only if all tasks are making progress. Consequently, if one of the participating nodes experiences a fault, the job is stalled while the task on that node resumes execution on that or another node. This amounts to assuming that the job experiences a fault each time one of its tasks experiences a fault, meaning that the jobs is paused during any task checkpointing, checkpoint recovery, or task migration.

With this tightly-coupled assumption, if faults are exponentially distributed, then $\mu_j = \mu/2^j$ (the minimum of n i.i.d exponential random variables with mean $1/\lambda$ is exponentially distributed with mean $1/(\lambda n)$). If faults follow a Weibull distribution with scale parameter a, then $\mu_j = \lambda\Gamma(1 + 1/(a2^j))$. Regardless, for all $0 \leq j \leq Z$, there are $\beta_j \times 2^j$ nodes each running one task of a 2^j-task job. We derive the throughput of preventive checkpointing for parallel workloads, ρ_{cp}, as

$$\rho_{cp} = \sum_{j=0}^{Z} \beta_j \times 2^j \times \max\left(0, \frac{\mu_j - R - C}{\mu_j + D}\right) . \quad (7)$$

For the preventive migration approach, the probability of encountering no shortage of spares is the same as in the sequential job case given in Equation 3. Indeed, the number of spares does not depend on the nature of the workload (sequential or parallel) but solely on the occurrences of failures. Consequently, the number of spares is the same whether the workload is sequential or parallel. Because there are only $N - n$ machines "really" available, we can simply scale the throughput by the factor $(N - n)/N$. We obtain the throughput of preventive migration for parallel workloads, ρ_{mp}, as

$$\rho_{mp} = \left(\sum_{j=0}^{Z} \beta_j \times 2^j \times \max\left(0, \frac{\mu_j - M}{\mu_j + D}\right)\right) \times \frac{N-n}{N} . \quad (8)$$

V. Numerical Results

While the main contribution of this work is the analytical models presented in the previous sections, here we present numerical results obtained using these models.

A. Model Instantiation

We present results obtained with our models for three scenarios (with C, R, D, and M in minutes):

- **Today**: $C = R = 10$, $D = 1$, and $M = 0.33$;
- **2011**: $C = R = 5$, $D = 1$, and $M = 0.33$;
- **2015**: $C = 0.21$, $R = 0.021$, $D = 0.25$, and $M = 0.33$.

It is impossible to pick unique and accurate values for these parameters, especially for future scenarios. However, we feel that the above is reasonably representative

of current and of projected future systems. For the values of C and R we assume local checkpoints on Hard Disk Drives (HDD) for the Today scenario, and on Solid State Drives (SSD) for the 2011 scenario. These values are based on our own measurements of checkpointing delays, assume 32GB of RAM today, and 64GB or RAM in 2011. Another possibility is to assume Flash memory disks in 2011, which improves over SSD by a factor 3.2 in our experiments. For simplicity, we assume that reading and writing times are identical, which is a coarse approximation. Regardless, our results for these two scenarios for parallel workloads show that preventive migration improve over preventive checkpointing by orders of magnitude in most cases. For the 2015 scenario, we assume the use of Phase-Change Memory (PCM) for the checkpointing "disk", whose cost is expected to be similar to that of DRAM before 2015. With this technology, the projected read bandwidth is 100GB/sec and the projected write bandwidth is 10GB/sec, hence the C and R values above assuming a 128GB checkpoint size.

For the reboot time, D, we assume 1 minute for the Today and the 2011 scenarios, and project 15 seconds for the 2015 scenario.

In all scenarios, the migration time, M, is set to 20 seconds. For current machines, this value is justified based on current performance evaluations of Infiniband technology [18] and on the configuration of current large-scale clusters. For the 2011 and 2015 scenarios, we project a doubling in bandwidth but, as mentioned earlier, also a doubling in node RAM as well, hence the constant M value for all three scenarios. In fact, the conclusions from our result do not change is M is increased to, say, 120 seconds, which is pessimistic.

For each scenario, we present results for sequential and parallel workloads, indicate the number of spare nodes needed for preventive migration, and use the following values for μ, N, and ε:

- $\mu = 1$ day, 1 week, 1 month, 1 year;
- $N = 2^{14}$, 2^{17}, 2^{20};
- $\varepsilon = 10^{-4}$, 10^{-6}.

B. Results and Discussion

Tables I to III show percentage improvement in expected throughput of the preventive migration approach over the preventive checkpointing approach (negative values indicate that preventive checkpointing is preferable).

For all scenarios, we see that both approaches have comparable (i.e., within 2%) performance for sequential workloads. For parallel workloads, instead, preventive checkpointing is outperformed by preventive migration by up to orders of magnitude in the Today and the 2011 scenarios. This is because of the high values of $C+R$ relatively to M in these scenarios and the fact that with parallel workloads a job is impacted by each fault occurred by each tasks. Large jobs end up paying the $C + R$ overhead very frequently. For the 2015 scenario, $C+R$ becomes dramatically lower when compared to the two previous scenarios. And, in fact, it becomes so low that preventive migration is outperformed by preventive checkpointing. However, the difference is at most on the order of 30% and checkpointing to a 128GB PCM disk corresponds to a configuration that may be deemed too expensive in the future, and may thus be overly optimistic for preventive checkpointing.

In all cases we note that the number of spares needed by preventive migration remains small (under 0.2% of the available nodes for a 1-day MTTF and a 2^{20}-node cluster). We also see that the results depend only slightly on ε. This is a good result for preventive migration and shows that one can easily enforce a very low probability of having to restart a jobs from scratch.

Our broad conclusion is that for the near-term future, preventive migration is definitely the approach of choice compared to preventive checkpointing. In the longer term, our results do not lead to a sharp differentiation between the two techniques and it is reasonable to expect comparable performance, with an advantage for preventive checkpointing.

Our implementation of the models is publicly available at http://navet.ics.hawaii.edu/~casanova/software/resilience.tgz.

VI. To Predict or Not To Predict

If no failure prediction is available, then one must use periodic checkpointing and rollback recovery. One interesting question, which has been studied for sequential jobs, is that of the frequency of checkpointing. In this section we develop a throughput model for the traditional periodic checkpointing and rollback recovery strategy, which we then contrast to preventive checkpointing and preventive migration.

A. Model

Let T be the checkpointing period, i.e., the time between two checkpoints. The expected percentage W of time lost, or "wasted", is

$$W = \frac{C}{T} + \frac{T}{2\mu} \; . \tag{9}$$

This expression is comes from the work in [19]. The first term is due to C seconds of checkpointing every T seconds. The second term accounts for the loss due to

Table I
PERCENTAGE IMPROVEMENT OF MIGRATION OVER CHECKPOINTING FOR SCENARIO **"TODAY"**. NUMBER OF REQUIRED SPARES IN PARENTHESES.

		Sequential Jobs		Parallel Jobs	
μ	N	$\varepsilon = 10^4$	$\varepsilon = 10^6$	$\varepsilon = 10^4$	$\varepsilon = 10^6$
1 day	2^{14}	1.19 (32)	1.16 (37)	3141.07 (32)	3140.08 (37)
	2^{17}	1.26 (164)	1.25 (177)	3086.92 (164)	3086.61 (177)
	2^{20}	1.28 (1086)	1.28 (1119)	3033.16 (1086)	3033.07 (1119)
1 week	2^{14}	0.14 (9)	0.12 (12)	3521.14 (9)	3520.47 (12)
	2^{17}	0.17 (35)	0.16 (40)	3511.74 (35)	3511.61 (40)
	2^{20}	0.18 (184)	0.18 (198)	3501.72 (184)	3501.67 (198)
1 month	2^{14}	0.02 (5)	0.00 (7)	1541.89 (5)	1541.69 (7)
	2^{17}	0.04 (13)	0.03 (17)	3354.95 (13)	3354.84 (17)
	2^{20}	0.04 (55)	0.04 (63)	3352.86 (55)	3352.83 (63)
1 year	2^{14}	-0.01 (2)	-0.01 (3)	69.22 (2)	69.21 (3)
	2^{17}	0.00 (4)	-0.00 (6)	1037.00 (4)	1036.99 (6)
	2^{20}	0.00 (11)	0.00 (13)	3381.52 (11)	3381.52 (13)

Table II
PERCENTAGE IMPROVEMENT OF MIGRATION OVER CHECKPOINTING FOR SCENARIO **"2011"**. NUMBER OF REQUIRED SPARES IN PARENTHESES.

		Sequential Jobs		Parallel Jobs	
μ	N	$\varepsilon = 10^4$	$\varepsilon = 10^6$	$\varepsilon = 10^4$	$\varepsilon = 10^6$
1 day	2^{14}	0.48 (32)	0.45 (37)	1587.29 (32)	1586.78 (37)
	2^{17}	0.55 (164)	0.54 (177)	1573.40 (164)	1573.24 (177)
	2^{20}	0.57 (1086)	0.57 (1119)	1558.96 (1086)	1558.91 (1119)
1 week	2^{14}	0.04 (9)	0.02 (12)	1743.11 (9)	1742.77 (12)
	2^{17}	0.07 (35)	0.07 (40)	1741.00 (35)	1740.93 (40)
	2^{20}	0.08 (184)	0.08 (198)	1738.54 (184)	1738.52 (198)
1 month	2^{14}	-0.01 (5)	-0.02 (7)	734.36 (5)	734.26 (7)
	2^{17}	0.01 (13)	0.01 (17)	1656.28 (13)	1656.23 (17)
	2^{20}	0.02 (55)	0.02 (63)	1655.80 (55)	1655.78 (63)
1 year	2^{14}	-0.01 (2)	-0.02 (3)	25.16 (2)	25.15 (3)
	2^{17}	-0.00 (4)	-0.00 (6)	477.62 (4)	477.61 (6)
	2^{20}	0.00 (11)	0.00 (13)	1668.73 (11)	1668.73 (13)

Table III
PERCENTAGE IMPROVEMENT OF MIGRATION OVER CHECKPOINTING FOR SCENARIO **"2015"**. NUMBER OF REQUIRED SPARES IN PARENTHESES.

		Sequential Jobs		Parallel Jobs	
μ	N	$\varepsilon = 10^4$	$\varepsilon = 10^6$	$\varepsilon = 10^4$	$\varepsilon = 10^6$
1 day	2^{14}	-0.12 (18)	-0.14 (22)	-27.96 (18)	-27.98 (22)
	2^{17}	-0.07 (82)	-0.08 (91)	-27.92 (82)	-27.92 (91)
	2^{20}	-0.05 (501)	-0.06 (523)	-27.90 (501)	-27.90 (523)
1 week	2^{14}	-0.04 (6)	-0.05 (8)	-13.14 (6)	-13.15 (8)
	2^{17}	-0.02 (20)	-0.02 (24)	-29.07 (20)	-29.08 (24)
	2^{20}	-0.01 (91)	-0.01 (101)	-29.07 (91)	-29.07 (101)
1 month	2^{14}	-0.02 (3)	-0.03 (5)	-2.63 (3)	-2.64 (5)
	2^{17}	-0.01 (8)	-0.01 (11)	-30.74 (8)	-30.74 (11)
	2^{20}	-0.00 (30)	-0.00 (35)	-30.74 (30)	-30.74 (35)
1 year	2^{14}	-0.01 (2)	-0.01 (2)	-0.22 (2)	-0.22 (2)
	2^{17}	-0.00 (3)	-0.00 (4)	-1.69 (3)	-1.69 (4)
	2^{20}	-0.00 (7)	-0.00 (9)	-17.00 (7)	-17.00 (9)

faults and is explained as follows: every μ seconds, a fault occurs and we lose an average of $T/2$ time-steps, which does not depend on the failure distribution. W is minimized for

$$T_{opt} = \sqrt{2C\mu}\ , \tag{10}$$

which is Young's approximation [20]. The corresponding minimum waste is

$$W_{min} = \sqrt{\frac{2C}{\mu}}\ . \tag{11}$$

However, this expression for W does not account for the recovery time R. A more accurate expression is

$$W = \frac{C}{T} + \frac{\frac{T}{2} + R + D}{\mu}\ .$$

This expression is minimized for the same value T_{opt} as before. However, the corresponding minimum waste becomes

$$W_{min} = \frac{R+D}{\mu} + \sqrt{2\frac{C}{\mu}}\ . \tag{12}$$

This values differs from the first-order approximation given by Daly [21], because we target the steady-state operation of the platform rather than the optimization of the expected duration of a given job. It turns out that W_{min} may become larger than 1 when μ becomes very small, a situation which is more likely to happen with jobs requiring many nodes in our model. In this case the application no longer makes progress. To solve for $W_{min} \leq 1$ in Equation 12, we let

$$\nu = \frac{1}{\sqrt{\mu}}\ , \tag{13}$$

and obtain

$$W_{min} \leq 1 \text{ if } \nu \leq \nu_b \Leftrightarrow \mu \geq 1/\nu_b^2 \tag{14}$$

with

$$\nu_b = \frac{-\sqrt{2C} + \sqrt{2C + 4(R+D)}}{2(R+D)}\ . \tag{15}$$

In all cases, the minimum waste is

$$W^*_{min} = \min(W_{min}, 1)\ . \tag{16}$$

For a sequential workload, we can now compute the expected throughput, ρ_f, as

$$\rho_f = (1 - W_{min})N\ . \tag{17}$$

For a parallel workload, using the parallel workload model and the notations described in Section IV, we compute $W_{min}(j)$ as in Equation 12, simply replacing μ by μ_j and letting

$$W^*_{min}(j) = \min(W_{min}(j), 1)\ . \tag{18}$$

We then obtain the throughput, ρ_{fp}, as

$$\rho_{fp} = \sum_{j=0}^{Z} (1 - W^*_{min}(j)) 2^j \beta_j\ . \tag{19}$$

B. Results and Discussion

Table IV shows ρ/N values in percentage, with 100% being optimal, for different values of N and two values of μ (1 month and 1 year), for a parallel workload in our 2015 scenario, for periodic checkpointing, preventive checkpointing, and preventive migration (with $\varepsilon = 10^{-6}$).

We see that fault avoidance, i.e., prediction-based approaches, achieves a significantly higher fraction of the optimal expected throughput than the fault tolerance approach at high scale. This is because fault tolerance is heavily impacted by the overhead of periodic checkpoints, during which a node is not performing useful work. This effect is compounded for parallel jobs. The use of non-blocking checkpointing, i.e., asynchronous checkpointing of a clone of the task, is possible in some cases and could lead to an improved throughput for fault tolerance. However, checkpointing the clone is not free and slows down the main process. Regardless, this result demonstrates the importance of fault prediction, which is necessary to achieve reasonable scaling of the platform beyond that achieved by fault tolerance.

In spite of this good relative result for failure avoidance, the best such approach for the 2015 scenario by a small margin according to our models, preventive checkpointing, achieves only a fraction of the optimal for large scale systems with 2^{20} nodes. With a long 1-year MTTF this fraction is below 55%, and with a short 1-month MTTF it is just above 5%. Such numbers are clearly not acceptable, especially because high utilization is key to justifying the startup and upkeep cost of a platform. Furthermore, recall that we we have made the optimistic assumption that failure prediction is perfectly accurate. We are thus considering a best-case scenario for fault avoidance.

The immediate, and alarming, conclusion from the above is that fault avoidance, although a good idea, is not sufficient for good utilization of future large-scale clusters. However, we note that our parallel job model assumes that some jobs can utilize the whole platform. In the next section we modify this assumption.

VII. Improving Utilization by Limiting Job Size

Given the assumption that parallel jobs are tightly coupled, and given the expected MTTFs for cluster nodes, jobs that use the platform at full scale are simply

	$\mu = 1$ month		
N	per. chkpt.	prev. chkpt.	prev. mig.
2^8	96.04%	99.81%	98.99%
2^{11}	88.23%	98.50%	98.04%
2^{14}	62.28%	88.75%	86.41%
2^{17}	10.66%	40.04%	27.73%
2^{20}	1.33%	5.01%	3.47%

	$\mu = 1$ year		
N	per. chkpt.	prev. chkpt.	prev. mig.
2^8	98.89%	99.98%	99.59%
2^{11}	96.80%	99.88%	99.75%
2^{14}	90.59%	99.01%	98.79%
2^{17}	70.46%	92.41%	90.84%
2^{20}	15.96%	54.77%	45.46%

Table IV
ρ/N FOR A PARALLEL WORKLOAD FOR SCENARIO **"2015"**, FOR PERIODIC CHECKPOINTING (PER.) AND PREDICTION-BASED PREVENTIVE CHECKPOINTING AND MIGRATION (PREV.).

	$\mu = 1$ month		
max job size	per. chkpt.	prev. chkpt.	prev. mig.
2^{20}	1.33%	5.01%	3.47%
2^{19}	2.67%	10.01%	6.93%
2^{18}	5.33%	20.02%	13.87%
2^{17}	10.66%	40.04%	27.73%
2^{16}	21.32%	63.07%	55.46%
2^{15}	42.64%	79.04%	74.72%

	$\mu = 1$ year		
max job size	per. chkpt.	prev. chkpt.	prev. mig.
2^{20}	15.96%	54.77%	45.65%
2^{19}	31.92%	73.57%	68.13%
2^{18}	55.59%	85.54%	82.56%
2^{17}	70.46%	92.41%	90.84%
2^{16}	80.05%	96.11%	95.30%
2^{15}	86.36%	98.03%	97.62%

Table V
ρ/N FOR A PARALLEL WORKLOAD FOR SCENARIO **"2015"** WITH CAPPED JOB SIZES, FOR PERIODIC CHECKPOINTING (PER.) AND PREDICTION-BASED PREVENTIVE CHECKPOINTING AND MIGRATION (PREV.).

not feasible when relying solely on the fault tolerance or fault avoidance techniques discussed thus far. For instance, with a MTTF of one year for each job, and exponentially distributed faults, a tightly coupled parallel job that utilizes 2^{20} nodes would experience a fault roughly every 0.5 minutes. This leads to a throughput that is close to or even equal to zero for fault tolerance or fault avoidance using the C, R, and M values in our 2015 scenario ($C = 0.21$, $R = 0.021$, $M = 0.33$). Equations 7, 8, and 19 give the expected value of the platform's throughput. The cases in which a job uses the whole platform is contributing to the expected value, thus contributing to the low values seen in Table IV. In this section we simply use our model to compute expected throughput values assuming that the job size is capped to a fraction of the platform. While the parallel job model in [17] does not consider such a cap, many existing system limit job sizes and such a limit is, as we just saw, sensible when faults are considered. The modification to our model is straightforward as long as the job size cap is a power of two: just replace Z by Z' in Equation 4, where the maximum jobs size is $2^{Z'}$.

We only present results for the 2015 scenario, for a 2^{20}-node cluster, for two values of μ (1 month and 1 year). Table V shows results for a parallel workload with different maximum job sizes from 2^{15} to 2^{20}, for periodic checkpointing, preventive checkpointing, and preventive migration (with $\varepsilon = 10^{-6}$).

These results show that, expectedly, capping the job

size leads to improved expected throughput. For the 1-month MTTF, the improvement is relatively slow. Even limiting job sizes to 2^{15} nodes, i.e., 1/32-th of the platform, preventive checkpointing does not quite achieve 80% utilization. However, with a 1-year MTTF, 85% utilization is achieved if jobs are restricted to at most 1/4-th of the platform.

We conclude that, unless additional mechanisms are used, high utilization for future large-scale platforms is only possible if job sizes are capped. Our model makes it possible to determine what this cap should be, based on the MTTF and the values of C, R, M, and D.

VIII. Conclusion

An alternative to traditional fault tolerance is failure avoidance, which is based on (accurate) fault prediction. We have derived models of the expected throughput of a large-scale cluster for preventive checkpointing and preventive migration. Our findings are as follows:

1) Preventive migration is preferable to preventive checkpointing for the near future by orders of magnitude;
2) For long-term scenarios, preventive checkpointing leads to improvements over preventive migration;
3) Expectedly, non-prediction-based traditional fault tolerance achieves very poor scaling for long-term scenarios and very large-scale platforms (i.e., 2^{20} nodes).
4) For these long-term scenarios and very large-scale platforms, even assuming perfect fault prediction, fault avoidance does not achieve good scaling (although it achieves scaling better than traditional fault tolerance).
5) At these large scales, it is necessary to cap the job size so that the platform can hope to achieve reasonable throughput, at least when the node MTTF is large enough (e.g., 1 year).

One interesting future direction for gaining further insight is the study of the impact of the accuracy of fault prediction on our results. In other words, what happens if the optimistic assumption that fault predictors are ideal is removed? New throughput models in this case, instantiated for statistical models of fault prediction accuracy, would allow a fair comparison with traditional fault tolerance. Regardless, our overall conclusion is that it is difficult to achieve large scale in future platforms by relying solely on fault avoidance. Unless new approaches are developed for deploying applications on future large-scale platforms, our results suggest that capping job sizes is required. However, even capping job sizes may not prove sufficient for low node MTTF (e.g., 1 month).

A future direction is the design of "self-fault-tolerant" algorithms (e.g., asynchronous iterative algorithms [22]) whose execution can make progress in the presence of local faults. Another possible direction is the use of task and/or job replication techniques. While replication can dramatically increase reliability, one concern is resource waste. Resource waste leads to extra energy consumption, which is a key concern for HPC installations. We expect that parallel jobs will be deployed successfully on large-scale machines only via the use of multiple techniques (checkpointing, migration, replication, self-tolerant algorithms).

Acknowledgment

Yves Robert is with the Institut Universitaire de France. This work was supported in part by the ANR StochaGrid and RESCUE projects, and by the INRIA-Illinois Joint Laboratory for Petascale Computing.

References

[1] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, A. Trefethen, and M. Valero, "The international exascale software project: a call to cooperative action by the global high-performance community," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 4, pp. 309–322, 2009.

[2] V. S. et al., "Exascale software study: Software challenges in extreme scale systems," *White paper. Only accessible on Internet*, 2009.

[3] "Top500 Supercomputer Sites," http://www.top500.org.

[4] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward Exascale Resilience," *Int. Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 374–388, 2009.

[5] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," *Journal of Physics: Conference Series*, vol. 78, no. 012022, 2007.

[6] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A meta-learning failure predictor for blue gene/l systems," *Parallel Processing, International Conference on*, vol. 0, p. 40, 2007.

[7] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive process-level live migration in hpc environments," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–12.

[8] S. Chakravorty, C. L. Mendes, and L. V. Kalé, "Proactive fault tolerance in mpi applications via task migration," in *HiPC 2006, the IEEE High performance Computing Conference*. IEEE Computer Society Press, 2006, pp. 485–496.

[9] N. Kolettis and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *FTCS '95: Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*. Washington, DC, USA: IEEE Computer Society, 1995, p. 381.

[10] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive management of software aging," *IBM J. Res. Dev.*, vol. 45, no. 2, pp. 311–332, 2001.

[11] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 575–584.

[12] D. G. Feitelson, "Memory usage in the LANL CM-5 workload," in *Proc. of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, April 1997, pp. 78–94.

[13] A. Batat and D. G. Feitelson, "Gang Scheduling with Memory Considerations," in *Proc. of the 14th International Parallel and Distributed Processing Symposium*, May 2000, pp. 109–114.

[14] S.-H. Chiang and M. K. Vernon, "Characteristics of a Large Shared-Memory Production Workload," in *Proc. of the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, June 2001, pp. 159–187.

[15] H. Li, D. Groep, and L. Wolters, "Workload Characteristics of a Multi-cluster Supercomputer," in *Proc. of the 10th International Workshop on Job Scheduling Strategies for Parallel Processing*, June 2004, pp. 176–193.

[16] S. K. Setia, M. S. Squillante, and V. K. Naik, "The impact of job memory requirements on gang-scheduling performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 4, pp. 30–39, 1999.

[17] U. Lublin and D. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs," *Journal of Parallel and Distributed Computing*, vol. 63, no. 11, pp. 1105–1122, 2003.

[18] M. J. Koop, W. Huang, K. Gopalakrishnan, and D. K. Panda, "Performance analysis and evaluation of pcie 2.0 and quad-data rate infiniband," in *HOTI '08: Proceedings of the 2008 16th IEEE Symposium on High Performance Interconnects*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 85–92.

[19] J. Wingstrom, "Overcoming The Difficulties Created By The Volatile Nature Of Desktop Grids Through Understanding, Prediction And Redundancy," Ph.D. dissertation, University of Hawai'i at Manoa, 2009.

[20] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Communications of the ACM*, vol. 17, no. 9, pp. 530–531, 1974.

[21] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303–312, 2004.

[22] A. Frommer and D. B. Szyld, "On asynchronous iterations," *J. Comput. Appl. Math.*, vol. 123, no. 1-2, pp. 201–216, 2000.