Preventive Migration vs. Preventive Checkpointing for Extreme Scale Supercomputers

Franck Cappello[*] Henri Casanova[†] Yves Robert[‡]

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 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 develop an analytical model of the performance for fault tolerance based on periodic checkpointing and compare this approach to both failure avoidance techniques. We find that this comparison is sensitive to the nature of the stochastic distribution of the time between failures, and that failure avoidance is likely inferior to fault tolerance in the long term. Regardless, our result show that each approach is likely to achieve poor utilization for large-scale platforms (e.g., 2^{20} nodes) unless the mean time between failures is large. We show how bounding parallel job size improves utilization, but conclude that achieving good utilization in future large-scale platforms will require a combination of techniques.

Keywords: failure prediction; checkpointing; migration; parallel jobs.

1 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 errors, arithmetic errors). 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. This makes it possible to recover from hardware faults, in case the faulty node needs to be replaced. On current large-scale clusters (i.e., those in the Top500 [3]), checkpointing 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

[*]INRIA-Illinois Joint Laboratory for Petascale Computing, Urbana-Champain, Illinois, U.S.A.
[†]Information and Computer Sciences Dept., University of Hawai‘i at Manoa, Honolulu, U.S.A.
[‡]Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon, France

checkpointing, will simply not scale sufficiently for future exascale machines. The alternative is to save checkpoint files on the local node, meaning that a task can resume execution only on the node on which the task was executing in the first place. With such local checkpoints, it is only possible to quickly recover from faults that come from software (but not from the application itself). Software faults can be resolved via software rejuvenation [6] before restarting the task. The state of a node after a rejuvenation is assumed to be free of software error, and rejuvenation is often simply achieved via rebooting [7].

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" [8]. 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* [9, 10]: 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. The faulty node can then be replaced, in case of a hardware fault, or software rejuvenation can be used in case of a software fault. 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 some of these limitations is *preventive checkpointing*: fault prediction is used to trigger a 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 is used before restarting the task on the same node. The main drawback of preventive checkpointing, when compared to preventive migration, is that execution is completely interrupted during a node reboot. Both preventive migration and preventive checkpointing are susceptible to false positives (predicted faults that do not occur), causing them to perform needless migration or checkpointing.

In this paper we compare preventive migration, preventive checkpointing, and traditional fault-tolerance in the context of HPC applications. Our main contributions are analytical performance models. We focus the comparison on software faults, meaning that all checkpoints are taken locally and that nodes can always return to a fault-free state via rejuvenation. Handling hardware faults would require the use of additional spare nodes for all our approaches, and we discount them in this work. Note that software faults account for between 40% and 75% of faults in HPC systems [5, 11].

This paper is organized as follows. Section 2 defines the problem, establishes notations, and introduces a performance metric. Section 3 provides analytical performance models for preventive checkpointing and preventive migration in the case of sequential workloads. Section 4 extends these models to the case of parallel workloads. Section 5 presents numerical results for current and likely future platform scenarios. Section 6 provides a comparison to non-prediction-based fault tolerance. Section 7 revisits results in Section 6 with the additional assumption that parallel job sizes are limited to a fraction of the cluster. Finally, Section 8 summarizes our main findings and gives future perspectives.

2 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 scheduling in our performance models. Each cluster node is subject to software failures. The time between failures (TBF) is a random variable with mean (MTBF) μ, in seconds. We leave the particular probability distribution of the TBF unspecified for now, but assume that failures are independent and identically distributed.

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. Unfortunately, no explicit analytical model to depict to depict task memory occupation is available in the literature. If 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, pessimistically, that 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. In practical current and foreseeable scenarios, however, $M < C + D + R$.

Our performance metric is the *expected steady-state throughput*, or *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 in the next two sections is to obtain analytical expressions for the throughput when migration- and checkpoint-based failure avoidance approaches are used. For the rest of the paper, we consider that faults do not happen during a checkpoint, a recovery or a migration.

3 Sequential Workloads

In this section we assume that each job uses only one node, and we estimate the throughput achieved by preventive checkpointing and by preventive migration.

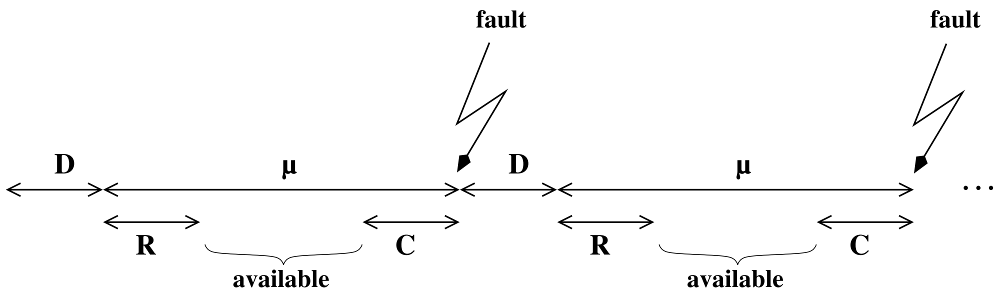

Figure 1: Execution time-line with preventive checkpointing.

3.1 Preventive Checkpointing

Let us compute the fraction of the time that a cluster node spends performing useful computation when preventive checkpointing is used, which we denote by f_c. Let $p(t)$ be the probability density function of the TBF, which has mean μ (the MTBF). f_c can be written as

$$f_c = \int_{R+C}^{\infty} \frac{t - R - C}{t + D} p(t) dt \; . \tag{1}$$

This equation is illustrated in Figure 1. A node goes through a sequence of time intervals: intervals of duration D during which the node is being rebooted, and availability intervals of duration t with probability $p(t)$, with mean μ, during which the node is available for computation before a fault occurs. Note that in the figure, for the sake of simplicity, it is assumed that all availability intervals are exactly of duration μ. At the beginning of each availability interval a checkpoint recovery, which takes take R seconds, must occur. At the end of each availability interval 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. Consequently, over an availability interval of duration t followed by a downtime period of duration D, for a total of $t + D$ seconds, a node performs $\max(0, t - R - C)$ seconds of useful computation. The maximum is to account for the fact that if $t \leq R + C$, then no useful computation can be performed. Equation 1 accounts for this maximum with the $R+C$ lower bound of the integral. Each of the N nodes in the cluster performs useful work on average a fraction f_c of the time. We conclude that the cluster throughput ρ_c, when preventive checkpointing is used, is obtained as

$$\rho_c = N \times f_c \; .$$

Computing the throughput then consists in computing f_c. The integral in Equation 1, however, may not be computable analytically. One can then consider two first-order approximations. The first approximation, which we denote by $f_c^{(1)}$, is given by

$$f_c^{(1)} = \frac{1}{\mu + D} \int_{R+C}^{\infty} (t - R - C) p(t) dt \; . \tag{2}$$

This approximation consists in dividing the expected duration of useful computation between a checkpoint and a checkpoint recovery by the MTBF. An even coarser approximation, $f_c^{(2)}$, is

$$f_c^{(2)} = \max\left(0, \frac{\mu - R - C}{\mu + D}\right) \; . \tag{3}$$

This last approximation can be computed analytically for any probability distribution function of the TBF, provided the MTBF is known. It would be useful to have a comparison between these approximations and the exact value, e.g., whether an approximation is pessimistic or optimistic in general. One can prove that $f_c^{(1)} \geq f_c^{(2)}$. In the case $\mu \leq R+C$, the proof is immediate since $f_c^{(2)} = 0$. When $\mu > R+C$, setting $A = R+C$ to shorten notations, one has:

$$\begin{aligned}
f_c^{(1)} &= \frac{1}{\mu+D}\int_{R+C}^{\infty}(t-A)p(t)dt \\
&= \frac{1}{\mu+D}\left[\int_A^{\infty} tp(t)dt - A\int_A^{\infty} p(t)dt\right] \\
&= \frac{1}{\mu+D}\left[\mu - \int_0^{A} tp(t)dt - A\int_A^{\infty} p(t)dt\right] \\
&= \frac{1}{\mu+D}\left[\mu - A\int_0^{A} \frac{t}{A}p(t)dt - A\int_A^{\infty} p(t)dt\right] \\
&\geq \frac{1}{\mu+D}\left[\mu - A\int_0^{A} p(t)dt - A\int_A^{\infty} p(t)dt\right] \\
&= \frac{\mu - A}{\mu+D} \;=\; f_c^{(2)} \;.
\end{aligned}$$

Unfortunately, no such result is available for comparing each approximation to the exact value. The approximations are pessimistic or optimistic depending on the nature of the underlying probability distribution and of the $R+C$ value.

In the following two sections, we compute f_c and $f_c^{(1)}$ for the Exponential and the Weibull distributions. The memoryless Exponential distribution is a common assumption when modeling TBF because it often lends itself to straightforward analytical derivations. The Weibull distribution is more complex but has been proposed as a more realistic alternative [12, 13].

3.1.1 Exponential Distribution

We consider an Exponential probability distribution with parameter $\lambda = 1/\mu$ and density function $p(t) = \lambda e^{-\lambda t}$. The $f_c^{(1)}$ approximation can be computed analytically by integrating by parts:

$$\begin{aligned}
f_c^{(1)} &= \frac{1}{\mu+D}\int_{R+C}^{\infty}(t-R-C)\lambda e^{-\lambda t}dt \\
&= \frac{\mu}{\mu+D}e^{-(R+C)/\mu}
\end{aligned}$$

Note that by replacing $e^{-(R+C)/\mu}$ by its first-order approximation $1-(R+C)/\mu$, assuming that $R+C \ll \mu$, one obtains $f_c^{(2)}$.

The exact f_c value is given by

$$f_c = \int_{R+C}^{\infty}\frac{t-R-C}{t+D}\lambda e^{-\lambda t}dt \;.$$

Let us define $X = R+C+D$, so as to simplify notations. A $u = t+D$ variable change and splitting into two terms gives

$$f_c = \lambda \int_X^\infty e^{-\lambda(u-D)} du \; - \lambda X \int_X^\infty \frac{e^{-\lambda(u-D)}}{u} du \; .$$

The first term can be easily integrated. A $v = \lambda u$ change in variable in the second term gives

$$f_c = e^{-\lambda(X-D)} - \lambda X e^{\lambda D} \int_{\lambda X}^\infty \frac{e^{-v}}{v} dv \; .$$

The "first order exponential integral", E_1, is classically defined as

$$E_1(x) = \int_1^\infty \frac{e^{-xt}}{t} dt = \int_x^\infty \frac{e^{-u}}{u} du \; .$$

We can thus write f_c in terms of E_1:

$$f_c = e^{-\lambda(X-D)} - \lambda X e^{\lambda D} E_1(\lambda X) \; .$$

or equivalently, with the original notations:

$$f_c = e^{-(R+C)/\mu} - \frac{R+C+D}{\mu} e^{D/\mu} E_1\left(\frac{R+C+D}{\mu}\right) \; .$$

There is no closed-form expression for E_1, but it can be be computed numerically with high accuracy. In this work we use the *gsl_sf_expint_E1* function of the GNU Scientific Library (GSL) to perform this computation.

3.1.2 Weibull Distribution

We consider a Weibull distribution with scale parameter λ and shape parameter a, with probability density $p(t) = (a/\lambda)(t/\lambda)^{a-1} e^{-(t/\lambda)^a}$. This distribution has mean $\mu = \lambda\Gamma(1 + 1/a)$, where $\Gamma(x)$ denotes the standard Gamma function. The $f_c^{(1)}$ approximation is then given by

$$f_c^{(1)} = \frac{1}{\mu + D} \int_{R+C}^\infty (t - R - C) \frac{a}{\lambda} (t/\lambda)^{a-1} e^{-(t/\lambda)^a} dt \; .$$

Integrating by part followed by a $u = (t/\lambda)^a$ variable change gives

$$\begin{aligned} f_c^{(1)} &= \frac{1}{\mu + D} \frac{\lambda}{a} \int_{((R+C)/\lambda)^a}^\infty u^{\frac{1}{a}-1} e^{-u} du. \\ &= \frac{1}{\mu + D} \frac{\lambda}{a} \, \Gamma(1/a, ((R+C)/\lambda)^a) \; . \end{aligned}$$

where $\Gamma(x, s)$ is the incomplete Gamma function defined as

$$\Gamma(x, s) = \int_s^\infty t^{x-1} e^{-t} dt \; .$$

There is no closed-form expression for the (incomplete) Gamma function, but it can be computed numerically with high accuracy. We use the *gsl_sf_gamma_inc* function of the GSL for this computation.

The exact f_c value is given by

$$f_c = \int_{R+C}^{\infty} \frac{t-R-C}{t+D} \frac{a}{\lambda} (t/\lambda)^{a-1} e^{-(t/\lambda)^a} dt \ .$$

Unfortunately, it does not seem to be possible to reduce this expression to known integrals. Consequently, one must resort to generic numerical integration, using the *gsl_integration_qagiu* function of the GSL. A $u = (t/\lambda)^a$ change of variable in the equation above is necessary so that the numerical integration does not lead to large errors (due to large values of λ).

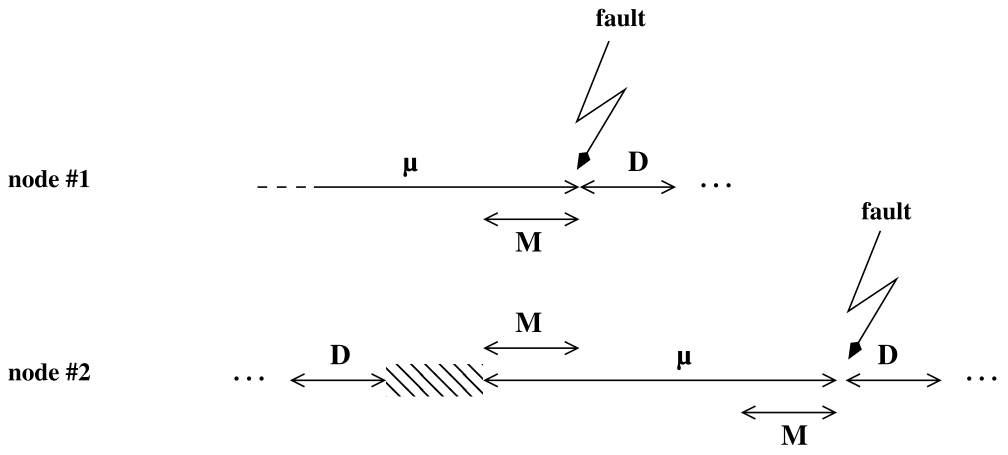

Figure 2: Execution time-line with preventive migration.

3.2 Preventive Migration

With preemptive migration a task must be migrated to a node that is immediately available. We call such a node a spare node, and we denote by n the number of spare nodes out of the N nodes available in the cluster. Importantly, these n spares are not fixed. After a failure a node becomes a spare node. Conversely, once a spare node starts being utilized to execute a task, it is no longer a spare node. This process is illustrated in Figure 2 for two nodes. Node 1 is involved in some computation. At the end of an interval of average duration μ, just before a failure, the computation is migrated to a spare node, which we call node 2 in the figure. Node 2 had been rebooted earlier (due to a failure that occurred while it was not a spare node). After its failure node 2 becomes a spare. It then reboots and waits idly (e.g., suspended) until a task needs to be migrated to it from a non-spare node, in our case node 1. After this migration, which has duration M, nodes 1 and 2 exchange their role. Node 1 becomes a spare node and is rebooted, while node 2 proceeds with the computation until the next failure. In steady-state, we see that there are two migrations during each availability interval of average duration μ for a given node: one at the beginning of the period, to receive data from the soon-to-fail node, and one at the end, to send data to a spare node just before failing. Following the same reasoning and notations as in the previous section, for an exponential distribution of parameter $\lambda = 1/\mu$, we can compute the fraction of the time that a non-spare node spends doing useful computation, f_m, as

$$f_m = \int_{2M}^{\infty} \frac{t-2M}{t-M} p(t) dt \ . \tag{4}$$

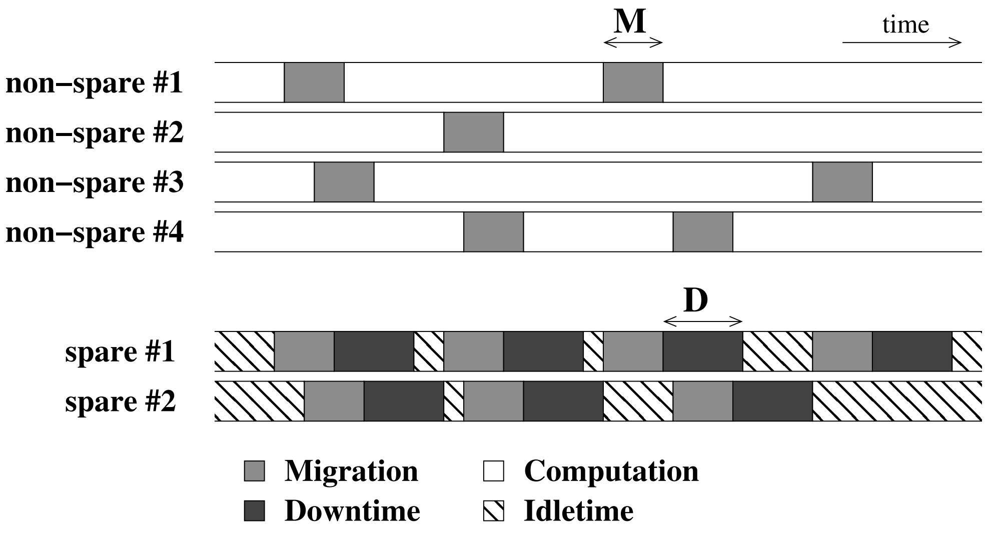

Figure 3: Sample execution with 4 non-spares and 2 spares.

Consider an availability interval of duration t. During the first $\max(0, t - M)$ seconds of each such interval, a non-spare node performs $\max(0, t - 2M)$ seconds of useful computations. These maxima are to account for the fact that if $t < 2M$ then no useful computation can be performed. Equation 4 accounts for these maxima with the $2M$ bound on the integral.

Altogether, there is a constant number $N - n$ of non-spare nodes, each of which performs useful computation a fraction f_m of the time. We thus obtain the cluster's throughput when preventive migration is used, ρ_m, as

$$\rho_m = (N - n) \times f_m \ .$$

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. We compute the probability of this event assuming that $\mu > M$, which is reasonable in practice. If $\mu < M$, then the probability is larger than the one computed hereafter.

We use Figure 3 for illustrative purposes. Recall that cluster nodes alternate between being spares and being non-spares. Each execution time-line in the figure corresponds to multiple physical cluster nodes that play the same role at different times throughout the execution. For each migration period (light gray box) for a non-spare, there is a corresponding migration period for a spare because a migration always occurs between a non-spare and a spare. Consider a period of arbitrary duration T. The expected time each non-spare spends migrating tasks during this period is $T \times \frac{M}{\mu - M}$. Indeed, each computation period (white box) has expected duration $\mu - 2M$, and is immediately followed by a migration period of duration M. The expected total time spent in migration by non-spares is $T \times (N - n) \times \frac{M}{\mu - M}$. The same amount of time must be spent in migration by spares, since a migration involves one spare and one non-spare. Furthermore, after spending M time units for a migration, a spare immediately spends D time units of downtime. Therefore, the expected total non-idle time spent by spares during the time period of duration T is $T \times (N - n) \times \frac{M+D}{\mu - M}$. Since there are n spares, the expected non-idle time spent by each spare is $T \times \frac{N-n}{n} \times \frac{M+D}{\mu - M}$. The probability that a spare node is not available (due to migration or downtime) at a given time is thus $\frac{N-n}{n} \times \frac{M+D}{\mu - M}$. We can now compute the probability that there is a shortage of spares at a given time, $P_{fail}(n)$, as the probability that each spare is not available. This is simply the probability that a spare is not available raised to the power n:

$$P_{fail}(n) = \left(\frac{N-n}{n} \times \frac{M+D}{\mu - M} \right)^n \ . \tag{5}$$

The goal is to choose n, the number of spares, so that $P_{fail}(n) \leq \varepsilon$, where ε is a low value specified by the cluster administrator. For a given ε, it is not possible to compute the corresponding n value analytically. However, since $P_{fail}(n)$ is monotonic, n can be computed numerically using a straightforward binary search.

It remains to compute f_m. Since Equation 4 is similar to Equation 1, we can obtain results similar to those in Section 3.1 for both the Exponential and the Weibull case. For completeness we list these results hereafter, where $f_m^{(1)}$ and $f_m^{(2)}$ denote the two approximations of f_m. These two approximations are valid only if $\mu > M$, which holds in practice. As in Section 3.1, one can show that $f_m^{(1)} \geq f_m^{(2)}$, but neither approximation is consistently optimistic or pessimistic when compared to the exact value.

3.2.1 Exponential Distribution

Considering an Exponential probability distribution with parameter $\lambda = 1/\mu$, we obtain

$$f_m^{(2)} = \frac{\mu - 2M}{\mu - M} \ ,$$

which is a closed form;

$$f_m^{(1)} = \frac{\mu}{\mu - M} e^{-2M/\mu} \ ,$$

which is a closed form; and

$$f_m = e^{-2M/\mu} - \frac{M}{\mu} e^{-M/\mu} E1(M/\mu) \ ,$$

where we compute the exponential integral E_1 numerically using the GSL function *gsl_sf_expint_E1*.

3.2.2 Weibull Distribution

Considering a Weibull probability distribution with scale parameter λ and shape parameter a, so that the mean is $\mu = \lambda\Gamma(1 + 1/a)$, we obtain

$$f_m^{(2)} = \frac{\mu - 2M}{\mu - M} \ ,$$

which is a closed form;

$$f_m^{(1)} = \frac{1}{\mu - M} \frac{\lambda}{a} \Gamma(1/a, (2M/\lambda)^a \ ,$$

where we compute the incomplete Gamma function numerically using the GSL function *gsl_sf_gamma_inc*; and

$$f_m = \int_{2M}^{\infty} \frac{t - 2M}{t - M} \frac{a}{\lambda} (t/\lambda)^{a-1} e^{-(t/\lambda)^a} dt \ ,$$

which we compute using the generic numerical integration GSL function *gsl_integration_qagiu*.

4 Parallel Workloads

4.1 Parallel Workload Model

The work in [14] proposes and validates models of parallel HPC workloads. 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 $\alpha = (1 - \alpha_0)/Z$.

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 $\beta_j = \alpha K$. 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:

$$\begin{aligned} N &= \sum_{j=0}^{Z} 2^j \alpha K \\ &= K \left(\alpha_0 + \frac{1-\alpha_0}{Z} \sum_{j=1}^{Z} 2^j \right) \\ &= K \left(\alpha_0 + \frac{1-\alpha_0}{Z} (2N-2) \right) . \qquad (6) \end{aligned}$$

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 $0 \leq j \leq Z$. 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 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. The entire job is paused unless all its tasks are running. With this tightly-coupled assumption, let us now consider a job that uses 2^j nodes. Let us denote by $p(t)$ the probability density function of the time to failure on a node. The TBF for a job using 2^j nodes is thus the minimum of 2^j i.i.d. random variables with probability density function $p(t)$. Let us use $p_j(t)$ to denote the probability density function of this minimum.

4.2 Preventive Checkpointing

Using the parallel workload model and the notations in the previous section, we derive the throughput when preventing checkpoint is used as follows. The throughput is the sum of the contributions of jobs that use 2^j nodes, for all $0 \leq j \leq Z$. At a given time, the expected number of nodes that execute tasks from jobs that use 2^j nodes is $\beta_j \times 2^j$. Let us define $f_{c,j}$ as the fraction of the time during which such a node performs useful computation. The contribution of all such nodes to the overall throughput is thus $\beta_j 2^j f_{c,j}$. We obtain the cluster's throughput when preventive checkpointing is used for parallel workloads, ρ_{cp}, by summing over j as

$$\rho_{cp} = \sum_{j=0}^{Z} \beta_j 2^j f_{c,j} \ , \tag{7}$$

where $f_{c,j}$ is given by Eq. 1 when replacing $p(t)$ by $p_j(t)$. We can now compute ρ_{cp} when node failures are Exponential or Weibull.

4.2.1 Exponential Distribution

The minimum of n i.i.d exponentially distributed variables with distribution parameter λ is also exponentially distributed with parameter $n\lambda$. Therefore, if node failures are exponentially distributed with parameter λ, $p_j(t)$ is the exponential distribution with parameter $\lambda_j = \lambda 2^j$ (and mean $\mu_j = 1/\lambda_j$). One can thus use the formulas in Section 3.1.1 simply replacing λ and μ by λ_j and μ_j.

4.2.2 Weibull Distribution

The minimum of n i.i.d. Weibull variables with shape parameter a and scale parameter λ is also Weibull with shape parameter a and scale parameter $\lambda/n^{1/a}$. Therefore, job failures are Weibull with scale parameter $\lambda_j = \lambda/2^{j/a}$, shape parameter a, and mean $\mu_j = \lambda 2^{-j/a}\Gamma(1+\frac{1}{a})$. One can thus use the formulas in Section 3.1.2 simply replacing λ and μ by λ_j and μ_j.

4.3 Preventive Migration

The throughput when preventive migration is used for parallel workloads can be computed following the same reasoning as for preventing checkpointing. Because there are only $N-n$ machines "really" available for computation, the throughput must be scaled by the factor $(N-n)/N$. Assuming exponentially distributed times to failure, we obtain the throughput of preventive migration for parallel workloads, ρ_{mp}, as

$$\rho_{mp} = \frac{N-n}{N} \sum_{j=0}^{Z} \beta_j 2^j f_{c,j} \ . \tag{8}$$

ρ_{mp}, or approximations of it, can be computed when node failures are Exponential or Weibull using the method explained in Section 4.2. Bounds on maximum relative errors are identical to those mentioned in that section as well.

The probability of encountering no shortage of spares is the same as in the sequential job case given in Equation 5. Indeed, the number of spares does not depend on the nature of the workload (sequential or parallel) but solely on the occurrences of failures at each node.

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

5.1 Model Instantiation

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

- **Local HDD (today)**: $C = R = 10$, $D = 1$, and $M = 0.33$;
- **Local SDD (2012)**: $C = R = 5$, $D = 1$, and $M = 0.33$;
- **Local Stable Memory (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 first scenario, and on SATA connected Solid State Drives (SSD) for the second scenario. These values are based on our own measurements of checkpointing delays, assume 32GB of RAM today (HDD scenario), and 64GB or RAM in 2012 (SDD scenario). For simplicity, we assume that reading and writing times are identical, which is a coarse approximation. Another possibility is to assume PCI connected SSDs. According to simple experiments we have conducted with both technologies, we have found that PCI connected SSDs improves over SATA SSD by a factor 3.2. Regardless, our results for these two scenarios for parallel workloads show that preventive migration improves over preventive checkpointing by orders of magnitude in most cases. For the Local Stable Memory scenario, we assume the use of Phase-Change Memory (PCM) for checkpointing, 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 (consistent with 2015).

For the reboot time D, we assume 1 minute for the two first scenarios, and project 15 seconds for the Local Stable Memory scenario (2015). 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 [15] and on the configuration of current large-scale clusters. For the SSD (2012) and Local Stable Memory (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 overall conclusions from our results 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:

- $\mu = 1$ month, 1 year, 10 years, 100 years, 1000 years;
- $N = 2^{14}$, 2^{17}, 2^{20};
- $\varepsilon = 10^{-6}$.

Regarding our choice of values for μ, there is no consensus in the literature regarding expected MTBFs, especially when considering software faults. Inspired by [5, 16], we have thus opted for choosing the above values as a way to explore what happens for different orders of magnitudes of the MTBF within a broad range, going to unlikely extremes. Note that a MTBF of 1 year per node would roughly correspond to that ASCI Q machine. The MTBF of 1000 years per node corresponds to the BlueGene design where all components of the system are selected and/or designed to match this objective. A MTBF of 10 to 100 years per node corresponds to supercomputers using more commodity components like the CRAY Jaguar. We also consider a MTBF of 1 month to evaluate how the comparison results evolve for extremely low MTBFs.

When using the Exponential distribution to model TBF, the distribution's parameter is obtained as $\lambda = 1/\mu$. When using the Weibull distribution, we set the distribution's shape parameter a to 0.78, using the value reported in [13]. The scale parameter is then computed numerically as $\lambda = \mu/\Gamma(1 + 1/a)$.

All results presented hereafter are for the exact throughput values derived in Sections 3 and 4 for sequential and parallel workloads, for Exponential and Weibull distributions of TBF. As explained in those sections, throughput computations rely on numerical approximations that we perform with GSL functions. All these functions return an upper bound on the absolute numerical error, which makes it possible to compute an upper bound on the maximum relative numerical error on the computed throughput. For all our results, all relative numerical errors on throughput values are below 0.006%. Also, recall that we assume a perfect prediction: no false positive nor false negative.

5.2 Results and Discussion

Tables 1 shows percentage improvement in throughput of the preventive migration approach over the preventive checkpointing approach (negative values indicate that preventive checkpointing is preferable). The top part of the table shows results assuming that the TBF follows an Exponential distribution, and the bottom part assumes a Weibull distribution. In both cases, results are presented for all values of μ and of N, for sequential and parallel workloads, and for our three scenarios.

For sequential workloads, we see that both approaches have comparable (i.e., within 3 points) performance across the board, regardless of the distribution of the TBF. The difference in performance is smallest for the 2015 scenario and for large values of μ. By contrast, for parallel workloads, preventive checkpointing is consistently outperformed by preventive migration in the Today and the 2011 scenarios, by up to several orders of magnitude. 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 of its 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. The difference is at most on the order of 50%. Furthermore, checkpointing to a 128GB PCM corresponds to a configuration that may be deemed too expensive in the future, and may thus be overly optimistic for preventive checkpointing. As expected, the difference between the two approaches decreases as μ increases.

The number of spares needed by preventive migration, which does not depend on the probability distribution, is under 0.07% of the total number of cluster nodes for all results in Table 1. The highest such fraction is for $N = 2^{14}$, for the Today scenario, with $\mu = 1$ week, in which case 10 spares are required. An interesting question is how this number of spares varies with ε (recall that our results are for $\varepsilon = 10^{-6}$). It turns out that the number of spares increases slowly as ε decreases. For instance, while in the above case 10 spares are needed for $\varepsilon = 10^{-6}$, only 15 spares are needed for $\varepsilon = 10^{-12}$. It is thus possible to enforce a low probability of running out of spares with a small number of spares.

Our broad conclusion is that for the near-term future preventive migration should be preferred over preventive checkpointing. In the longer term, however, both techniques should lead to comparable performance, with perhaps a small advantage for preventive checkpointing.

Table 1: Percentage improvement of migration over checkpointing.

			Sequential Jobs			Parallel Jobs		
	μ	N	**Today**	**2012**	**2015**	**Today**	**2012**	**2015**
Exponential	1 week	2^{14}	1.28	0.69	-0.03	3169.61	1566.90	-21.89
		2^{17}	1.32	0.73	-0.00	3162.23	1565.35	-50.64
		2^{20}	1.33	0.73	0.00	3153.87	1563.29	-50.72
	1 month	2^{14}	0.34	0.17	-0.03	1416.99	671.71	-4.77
		2^{17}	0.37	0.20	-0.00	3119.09	1537.22	-41.12
		2^{20}	0.38	0.21	0.00	3117.25	1536.80	-51.74
	1 year	2^{14}	0.02	-0.00	-0.02	125.13	63.86	0.11
		2^{17}	0.04	0.02	-0.00	1001.34	463.32	-2.65
		2^{20}	0.04	0.02	-0.00	3368.85	1662.41	-25.43
	10 years	2^{14}	-0.01	-0.02	-0.01	16.53	9.29	0.07
		2^{17}	0.00	-0.00	-0.00	101.11	52.55	0.15
		2^{20}	0.00	0.00	-0.00	817.80	374.43	-1.88
	100 years	2^{14}	-0.01	-0.01	-0.01	2.43	1.36	0.00
		2^{17}	-0.00	-0.00	-0.00	13.76	7.75	0.07
		2^{20}	0.00	0.00	-0.00	82.10	43.36	0.16
	1000 years	2^{14}	-0.01	-0.01	-0.01	0.33	0.17	-0.01
		2^{17}	-0.00	-0.00	-0.00	2.03	1.13	0.01
		2^{20}	-0.00	-0.00	-0.00	11.44	6.46	0.06
Weibull	1 week	2^{14}	2.88	1.70	-0.03	1577.96	901.29	-33.95
		2^{17}	2.92	1.74	-0.00	1562.56	896.12	-33.93
		2^{20}	2.92	1.74	0.00	1546.95	890.71	-33.91
	1 month	2^{14}	0.98	0.57	-0.03	1603.11	908.94	-31.00
		2^{17}	1.01	0.60	-0.00	1602.61	910.05	-33.96
		2^{20}	1.01	0.60	0.00	1597.24	908.25	-33.95
	1 year	2^{14}	0.13	0.07	-0.02	903.16	492.74	-7.31
		2^{17}	0.15	0.09	-0.00	1618.12	914.81	-31.93
		2^{20}	0.16	0.09	-0.00	1619.33	915.73	-33.97
	10 years	2^{14}	0.01	-0.00	-0.01	174.78	95.24	-0.64
		2^{17}	0.02	0.01	-0.00	1076.58	594.55	-9.92
		2^{20}	0.03	0.02	-0.00	1622.29	916.66	-33.24
	100 years	2^{14}	-0.01	-0.01	-0.01	30.84	18.36	-0.00
		2^{17}	0.00	0.00	-0.00	233.99	125.24	-0.98
		2^{20}	0.00	0.00	-0.00	1242.25	692.23	-13.13
	1000 years	2^{14}	-0.01	-0.01	-0.01	6.09	3.66	0.00
		2^{17}	-0.00	-0.00	-0.00	40.05	23.66	-0.01
		2^{20}	0.00	0.00	-0.00	312.51	165.03	-1.50

6 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. 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. In this section we only consider blocking checkpointing.

6.1 Model

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

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

This expression comes from the work in [17]. The first term is due to C seconds of checkpointing every T seconds. The second term accounts for the loss due to 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 TBF distribution. W is minimized for

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

which is known as Young's approximation [18]. The corresponding minimum waste is

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

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

$$W_{min} = \frac{R+D}{\mu} + \sqrt{2\frac{C}{\mu}} \ . \tag{9}$$

This values differs from the first-order approximation given by Daly [19], because we target the steady-state operation of the cluster 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 9, we let

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

and obtain

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

with

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

In all cases, the minimum waste is

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

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

$$\rho_f = (1 - W^*_{min})N \; .$$

For a parallel workload, using the parallel workload model and the notations in Section 4.1, we compute $W_{min}(j)$ as in Equation 9, simply replacing μ by μ_j and letting

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

Following the same reasoning as in Sections 4.2 and 4.3, we obtain the throughput, ρ_{fp}, as

$$\rho_{fp} = \sum_{j=0}^{Z} \beta_j \times 2^j \times (1 - W^*_{min}(j)) \; . \tag{10}$$

6.2 Results and Discussion

Table 2 shows ρ/N values in percentage, for different values of N, three values of μ (1 month, 1 year, and 10 years), for a parallel workload, for the 2015 scenario. We call this value the *yield*, with 100% being optimal. Results are shown for periodic checkpointing, preventive checkpointing, and preventive migration (with $\varepsilon = 10^{-6}$). For the two preventive approaches, results are shown both for an Exponential and a Weibull distribution of the TBF.

Given that the results are for the 2015 scenario, the yield of preventive checkpointing is higher than for preventive migration (see Section 5.2). As expected, the yield decreases as μ decreases or N increases. Considering results assuming an Exponential distribution of TBF, failure avoidance provides some improvement over fault tolerance, by up to about 25 points. However, results assuming a Weibull distribution show a very different picture. In this case, the yield achieved by failure avoidance is markedly lower than that of fault tolerance. The explanation for this result is that the shape parameter of the Weibull distribution, $a = 0.78$, is strictly lower than 1. This value corresponds to decreasing failure rates as reported in [13]. The probability of having small TBF values is higher, leading to more time spent doing task migrations. In fact, results for a Weibull distribution can only improve upon results for an Exponential distribution if $a > 1$. However, results based on analyzing failure logs report values strictly lower than 1 [12, 13], thus further confirming that assuming exponentially distributed TBF is not realistic. Interestingly, the yield achieved by preventive checkpointing would be higher than that achieved by periodic checkpointing in all cases if a were to be set to 0.94. Consequently, within the confines of our model, even when assuming a Weibull distribution with decreasing failure rates, it is possible for failure avoidance to outperform fault tolerance.

			Exponential		Weibull	
	N	per. chkpt.	prev. chkpt.	prev. mig.	prev. chkpt.	prev. mig.
μ =1 week	2^8	91.56%	96.28%	95.30%	83.71%	81.18%
	2^{11}	73.75%	82.95%	81.19%	42.92%	34.67%
	2^{14}	20.07%	46.03%	35.95%	7.30%	4.82%
	2^{17}	2.51%	9.11%	4.50%	0.91%	0.60%
	2^{20}	0.31%	1.14%	0.56%	0.11%	0.08%
μ =1 month	2^8	96.04%	98.86%	97.81%	93.35%	91.97%
	2^{11}	88.23%	93.97%	93.87%	68.83%	64.41%
	2^{14}	62.28%	74.64%	71.08%	21.67%	14.96%
	2^{17}	10.66%	32.04%	18.87%	2.84%	1.88%
	2^{20}	1.33%	4.89%	2.36%	0.36%	0.23%
μ =1 year	2^8	98.89%	99.87%	99.11%	98.80%	98.05%
	2^{11}	96.80%	99.20%	99.14%	92.58%	92.09%
	2^{14}	90.59%	95.63%	95.74%	66.21%	61.37%
	2^{17}	70.46%	80.49%	78.36%	19.31%	13.14%
	2^{20}	15.96%	41.38%	30.85%	2.49%	1.65%
μ =10 years	2^8	99.65%	99.98%	99.21%	99.78%	99.01%
	2^{11}	99.00%	99.89%	99.81%	98.45%	98.37%
	2^{14}	97.15%	99.34%	99.40%	90.66%	90.08%
	2^{17}	91.63%	96.33%	96.48%	60.12%	54.16%
	2^{20}	74.01%	83.15%	81.59%	14.84%	9.91%
μ =100 years	2^8	99.89%	100.00%	99.22%	99.96%	99.18%
	2^{11}	99.69%	99.99%	99.89%	99.71%	99.62%
	2^{14}	99.11%	99.91%	99.91%	97.99%	97.99%
	2^{17}	97.45%	99.45%	99.52%	88.31%	87.44%
	2^{20}	92.56%	96.93%	97.08%	53.56%	46.53%
μ =1000 years	2^8	99.97%	100.00%	99.61%	99.99%	99.60%
	2^{11}	99.90%	100.00%	99.90%	99.95%	99.85%
	2^{14}	99.72%	99.99%	99.98%	99.62%	99.62%
	2^{17}	99.20%	99.93%	99.94%	97.41%	97.41%
	2^{20}	97.73%	99.55%	99.61%	85.46%	84.18%

Table 2: ρ/N for a parallel workload for scenario **"2015"**, for periodic checkpointing (per.) and prediction-based preventive checkpointing and migration (prev.).

Discounting the results for the Exponential distribution, and assuming that $a = 0.78$ is representative of future systems, our overall conclusion is that failure avoidance provides lower yield than periodic checkpointing. Furthermore, note that for failure avoidance, we have made the optimistic assumption that fault predictions are perfectly accurate. In reality, there will be false negatives, and failure avoidance can only be used in practice in conjunction with some form of periodic checkpointing.

Regardless of how the different approaches fare, one concern is that the achieved yield is low for large scale systems (i.e, 2^{20} nodes) unless μ is very large (i.e., 10 years). For instance, once again discounting the results for the Exponential distribution, for a 2^{20}-node cluster and a large μ of 1 year, fault-tolerance achieves a yield below 16% and fault-avoidance below 3%. Such numbers are simply not acceptable, especially because high utilization is key to justifying the startup and operational cost of a cluster. The immediate, and alarming, conclusion from the above is that neither fault tolerance or failure avoidance are sufficient for good utilization of future large-scale clusters (except maybe for a MTBF per node close to 1000 years, a value which may be extremely difficult to achieve in 2015 and after). A reason for this result is that our parallel job model assumes that some jobs can utilize the whole cluster. In the next section we modify this assumption.

7 Improving Yield by Limiting Job Size

Given the assumption that parallel jobs are tightly coupled, jobs that use the cluster at full scale are simply not feasible when relying solely on the fault tolerance or failure avoidance techniques discussed thus far. For instance, with $\mu = 1$ year and exponentially distributed faults, a tightly coupled parallel job that utilizes 2^{20} nodes would experience a fault roughly every 0.5 minute. This leads to a throughput that is close to or even equal to zero for fault tolerance or failure 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 10 give the expected value of the cluster's throughput. The cases in which a job uses the whole cluster is contributing to the expected value, thus contributing to the low values seen in Table 2. In this section we use our model to compute throughput values assuming that the job size is capped to a fraction of the cluster. While the parallel job model in [14] 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 6, where the maximum jobs size is $2^{Z'}$.

Table 3 presents results for the 2015 scenario, for a 2^{20}-node cluster, for a parallel workload with maximum job sizes ranging from 2^{15} to 2^{20}. Results are shown for periodic checkpointing, preventive checkpointing, and preventive migration (with $\varepsilon = 10^{-6}$). The conclusions regarding the relative merit of the three approaches are unchanged when compared to the discussion in Section 6.2. Expectedly, capping the job size leads to improved throughput. For $\mu = 1$ month, the improvement is relatively slow. Even limiting job sizes to 2^{15} nodes, i.e., 1/32-th of the cluster, no approach achieves higher than a 50% yield, unless an Exponential distribution is assumed. However, with $\mu = 1$ year, a yield of more than 85% is achieved with periodic checkpointing if jobs are restricted to using at most 1/16-th of the cluster. This yield reaches almost 95% for $\mu = 10$ years.

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

			Exponential		Weibull	
	job cap	per. chkpt.	prev. chkpt.	prev. mig.	prev. chkpt.	prev. mig.
μ =1 week	2^{20}	0.31%	1.14%	0.56%	0.11%	0.08%
	2^{19}	0.63%	2.28%	1.12%	0.23%	0.15%
	2^{18}	1.25%	4.56%	2.25%	0.46%	0.30%
	2^{17}	2.51%	9.11%	4.50%	0.91%	0.60%
	2^{16}	5.02%	17.60%	8.99%	1.83%	1.21%
	2^{15}	10.04%	30.57%	17.98%	3.65%	2.41%
μ =1 month	2^{20}	1.33%	4.89%	2.36%	0.36%	0.23%
	2^{19}	2.67%	9.75%	4.72%	0.71%	0.47%
	2^{18}	5.33%	18.70%	9.43%	1.42%	0.94%
	2^{17}	10.66%	32.04%	18.87%	2.84%	1.88%
	2^{16}	21.32%	47.58%	37.74%	5.68%	3.75%
	2^{15}	42.64%	62.41%	55.69%	11.32%	7.50%
μ =1 year	2^{20}	15.96%	41.38%	30.85%	2.49%	1.65%
	2^{19}	31.92%	56.79%	48.71%	4.98%	3.29%
	2^{18}	55.59%	70.18%	65.49%	9.95%	6.58%
	2^{17}	70.46%	80.49%	78.36%	19.31%	13.15%
	2^{16}	80.05%	87.78%	87.06%	33.68%	25.40%
	2^{15}	86.36%	92.59%	92.51%	50.52%	43.08%
μ =10 years	2^{20}	74.01%	83.15%	81.59%	14.84%	9.91%
	2^{19}	82.37%	89.56%	89.12%	27.30%	19.59%
	2^{18}	87.90%	93.73%	93.75%	43.58%	35.45%
	2^{17}	91.63%	96.33%	96.48%	60.12%	54.16%
	2^{16}	94.18%	97.89%	98.04%	73.95%	70.63%
	2^{15}	95.93%	98.81%	98.93%	84.02%	82.51%
μ =100 years	2^{20}	92.56%	96.93%	97.08%	53.56%	46.53%
	2^{19}	94.81%	98.24%	98.39%	68.73%	64.39%
	2^{18}	96.37%	99.01%	99.12%	80.36%	78.23%
	2^{17}	97.45%	99.45%	99.52%	88.31%	87.44%
	2^{16}	98.21%	99.70%	99.74%	93.32%	93.03%
	2^{15}	98.74%	99.84%	99.86%	96.29%	96.23%
μ =1000 years	2^{20}	97.73%	99.55%	99.61%	85.46%	84.18%
	2^{19}	98.40%	99.75%	99.79%	91.56%	91.10%
	2^{18}	98.87%	99.86%	99.89%	95.27%	95.14%
	2^{17}	99.20%	99.93%	99.94%	97.41%	97.41%
	2^{16}	99.44%	99.96%	99.97%	98.61%	98.64%
	2^{15}	99.60%	99.98%	99.98%	99.27%	99.29%

Table 3: 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.).

8 Conclusion

An alternative to traditional fault tolerance is failure avoidance, which is based on fault prediction. We have derived models of the expected steady-state 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, but preventive checkpointing will likely lead to some improvements over preventive migration in long-term scenarios. Our models can be used to quantify the relative performance of both approaches given system parameters.

2. In long-term scenarios, if one assumes a Weibull distribution with shape parameter below $a = 0.94$ (results in [12] point to $a = 0.78$), then failure avoidance leads to lower utilization than fault tolerance. The traditional Exponential assumption points to significant advantage of failure avoidance over fault tolerance and likely does not hold in practice.

3. For large-scale platforms (e.g., 2^{20} nodes), all approaches lead to unacceptably low platform utilization unless the MTBF is very large. At such scales, it is necessary to cap the job size so that the cluster can hope to achieve reasonable throughput. Our models provide a way to determine this cap.

Note that in this work we have assumed system-level checkpointing, meaning that the checkpointing period and the checkpoint data are not chosen to exploit particular application features. By contrast, in application-level checkpointing a checkpoint can be taken at particular times (e.g., at the end of each application iteration) and for particular data (e.g., only those data that are needed to preserver an application-defined state). If application-level checkpointing were available, then the checkpointing overhead could be reduced significantly, thus changing our results in favor of fault tolerance implemented with periodic checkpointing. Unfortunately, application-level checkpointing is not wide-spread, partly due to the fact that it had to be implemented manually for each given application.

One possible future direction 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 not possible to achieve large scale in future clusters by relying solely on failure avoidance. Consequently, removing the optimistic assumption would simply strengthen this negative result. Unless new approaches are developed for deploying applications on future large-scale clusters, our results suggest that capping job sizes is required. However, even capping job sizes may not prove sufficient for low node MTBF. Finally, not that our results discount hardware faults, which further limit the throughput achievable by both fault tolerance and failure avoidance.

A future direction is the design of "self-fault-tolerant" algorithms (e.g., asynchronous iterative algorithms [20]) 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 and diminished throughput. 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. Sarkar and others, "Exascale software study: Software challenges in extreme scale systems," 2009, white paper. Only accessible on Internet at http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf.

[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] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[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] T. Heath, R. P. Martin, and T. D. Nguyen, "Improving cluster availability using workstation validation," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 217–227, 2002.

[13] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proc. of the International Conference on Dependable Systems and Networks*, 2006, pp. 249–258.

[14] 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.

[15] 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.

[16] L. Wang, P. Karthik, Z. Kalbarczyk, R. Iyer, L. Votta, C. Vick, and A. Wood, "Modeling Coordinated Checkpointing for Large-Scale Supercomputers," in *Proc. of the International Conference on Dependable Systems and Networks*, June 2005, pp. 812–821.

[17] 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.

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

[19] 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.

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