Performance Analysis of Distributed Systems that Perform \( N \) Tasks using \( P \) Fault-Prone Processors without Repair

Gehan Weerasinghe and Lester Lipsky
Department of Computer Science and Engineering
and
The Taylor L. Booth Center for Computer Applications and Research
University of Connecticut, Storrs, Connecticut 06269-3031, U.S.A
\{weerasin, lester\}@engr.uconn.edu

July 19, 2002

Abstract

This paper presents a family of Markov models for analyzing the performance of parallel/distributed systems that execute a job consisting of \( N \) independent tasks using \( P \) processors in parallel. The model is a Markov Chain with states representing service and failure rates with \( k \) (\( 0 < k \leq P \)) active processors. The task-times and processor failures are both exponentially distributed. A number of formulas/algorithms are derived for determining the probability of system failure, average number of processor failures, failure distribution and mean time to failure, mean execution time, work, and other measurable quantities. Since the set of tasks to be processed is fixed (i.e. there is no arrival process), and there is no repair process, there is a finite probability that the job will never finish. Therefore, the performance parameters must be conditioned on the job finishing successfully. Some results are presented and analyzed for a range of values of processor failure rates, where \( P << N \) and also where \( P = N \).
# Contents

1 Introduction and Motivation  ......................................................... 3  
1.1 Related Work ........................................................................ 4  
1.2 Contributions ....................................................................... 5  

2 Background: Processors With No Failures  ...................................... 5  

3 Processors with Failures: The Base Model  ...................................... 8  
3.1 Mathematical Preliminaries ...................................................... 9  
3.2 Base Model ........................................................................... 10  
3.2.1 Probability of Job Failure ................................................... 10  
3.2.2 Periodicity Parameter ........................................................... 12  
3.2.3 Average Number of Processor Failures ................................. 13  
3.2.4 Average Number of Completed Tasks Before Failure ........... 14  
3.2.5 Time for Job to Finish .......................................................... 15  
3.2.6 Mean Time To Failure .......................................................... 16  
3.2.7 The Work Function ............................................................... 16  
3.2.8 Retraining of Failed Tasks .................................................... 17  

4 Generalized Model .......................................................................... 18  
4.1 Formulas for the Generalized Model .......................................... 19  
4.2 Case 1: All Functioning Processors can fail .............................. 19  
4.3 Case 2: Failure-Free Processor, Idle Processors do not fail ........ 19  
4.4 Case 3: Failure-Free Processor, Idle Processors Can Fail .......... 20  
4.5 Case 4: Last Processor can be Repaired ..................................... 21  
4.6 Case 5: Other Possibilities ....................................................... 21  

5 Some Calculations of the Base Model \((N \gg P)\)  ................................. 21  
5.1 Failure Probability ................................................................... 22  
5.2 Mean Number of Failed Processors .......................................... 22  
5.3 Time of Job Execution ............................................................... 23  
5.4 Work ..................................................................................... 24  

6 Results for the Generalized Model .................................................. 24  
6.1 Results for the General Case for \(N \gg P\) .................................... 26  
6.2 Results for \(N = P\) ................................................................. 27  

7 Conclusion .................................................................................... 30
1 Introduction and Motivation

The use of Networks of Workstations (NoWs) for high performance computing [BUY99, PFIS98] has become a cost effective alternative to buying a supercomputer. To fully utilize NoWs, these applications should have the ability of tolerating permanent processor failures, communication time-outs etc. In the presence of such failures, applications should continue execution and produce correct results. Thus, it is a formidable task to design and analyze fault tolerant distributed systems and applications. Therefore, it is important to devise a model for analyzing the performance of such systems that would eventually lead to providing a better understanding of the design considerations and practicality in developing distributed systems.

When constructing an analytic performance model, it is imperative that one focus on a particular application domain. In the field of high performance scientific computing, applications that are based on processing a bag of tasks form a large domain [BAKK95, BARA99, WEER01c]. These applications are also referred to as iterative, grid and data parallel. Some examples are: Simulation, Image Processing, Discrete Optimization, Transformation, and Computational Geometry. A task can easily be defined in these applications. For example, in image processing, classifying a pixel or a set of pixels can be considered as a task, and classifying the whole image as a job. Most of these applications can be parallelized as long as the defined tasks are both independent and idempotent. This paper focuses on this application domain. Thus, the performance of systems that execute a job consisting of $N$ independent tasks using $P$ fault-prone processors in parallel is analyzed.

This paper presents a family of Markov Models for analyzing the performance of the above application domain. These analytic models are based on Markov Chains with states representing service and failure rates with $k$ ($0 < k \leq P$) active processors. It is assumed that the task-times and processor failures are both exponentially distributed, all processors are identical and the probability rate of failure for each processor is constant throughout the execution. A number of expressions are derived to determine the mean execution time, probability of system failure, work, and other measurable quantities, all conditioned on the job finishing successfully. The input parameters are $N$, $P$ and $\tau$ and the processor failure rate is $\alpha$.

The models assume that if a processor fails while executing a task the task later restarts where it left off. Since it is assumed here that the task times are exponentially distributed, this turns out to be equivalent to having a new task taken from the same distribution start from the beginning. (For non-exponential task times, these would not be equivalent.) In certain environments, if a task fails, it must start over again later, and it must run at least as long as it did previously before it failed. This discrepancy is examined in Section 3.2.8.

The models presented here evaluate the performance of systems that execute a job consisting of $N$ tasks using $P$ fault-prone processors for the following five cases: case 0) only busy processors can fail (this is the base model); case 1) all processors can fail; case 2) a fault-free processor exists, and idle processors cannot fail; case 3) a fault-free processor exists, and idle processors can fail; case 4) if the last processor fails before job completion, it is repaired. Other scenarios are also discussed. Preliminary results were presented
in [WEER01b] and [WEER02a], where they were also compared with actual experiments ([WEER01c]). In this paper, we also examine the results for scaled behavior. Such analysis can give insight into how systems behave for different values of system parameters. For instance, we found that a system with $P$ processors executing $N$ tasks with a failure rate of $\alpha$ will have almost the same probability of failure as a $P$ processor system executing $2N$ tasks with a failure rate of $\alpha/2$ (probability scales well). However, the conditional mean execution time and work done do not scale accordingly. That is, in general for failure probability, $\mathcal{F}(N_1, P, \gamma/N_1) \approx \mathcal{F}(N_2, P, \gamma/N_2)$, but $(1/N_1)T(N_1, P, \gamma/N_1) \neq (1/N_2)T(N_2, P, \gamma/N_2)$.

1.1 Related Work

The subject of system reliability, Performability and performance analysis of parallel and distributed systems and algorithms have a long and a successful history, and is a large component of many texts and articles [FELL71, GAY79, MEYE80, LIPS92, LIPS96, SAHN96, TRIV82] etc. Many models of fault-tolerant distributed systems assume a Poisson arrival of independent tasks/jobs (such as in an M/G/P queue) and look at steady-state behavior [DONA87, HARI95, MOSS94]. Various characteristics such as processor failures and repair are modeled using Markov chains with states representing the system’s status. In such environments, the systems are idle a fraction of the time. We focus on a fixed set of tasks (a job) and evaluate the performance of the distributed system executing these tasks. Hence, our model is transient. During execution, a system may have fewer tasks remaining than the number of processors still operative (i.e., some processors are idle and serve as backup). Therefore we analyze each system for two alternative cases, one of which allows idle processors to fail, and the other does not.

A widely used performance metric is Performability [MEYE80]. It uses both performance and reliability issues to analyze the performance of degradable systems. Most of the studies that incorporate performability use reward or profit [DONA87, NABL96, WOLT00] models in which a reward rate is specified for the work accomplished in a specific state. Then the performance of a system is determined by the total accumulated reward for a specified time period. Thus one can capture the effective performance of a system, its minimum reliability [SAHN96] and optimal utilization of available system resources [BOND97].

The use of performability metrics to evaluate a real-time fault-tolerant system through determining the redundancy levels of tasks is presented in e.g. [WANG95, GONZ97].

Our strategy in evaluating the performance of a fault-tolerant system differs from the aforementioned studies as follows. First, we analyze the performance of processing a job consisting of a fixed set of $N$ tasks using $P$ processors in parallel. Therefore, there is no arrival process for tasks and hence the performance model is transient. Since the task times are non-deterministic, we don’t address the real-time issue of meeting a deadline. We also assume that when there are more processors available than there are tasks remaining, the excess processors sit idle, and pick up a task only if an active processor fails. Furthermore, since the task times are unknown at execution, workload is balanced dynamically and hence a schedule-based analysis is irrelevant.
1.2 Contributions

1. We have devised a family of models that can be used to analyze a system that performs a job consisting of $N$ independent tasks using $P$ processors that can fail.

2. We have derived expressions to determine the following system performance characteristics, where the last four are conditioned on the job finishing successfully:
   - The Probability that the System will Fail.
   - Failure Distribution and Mean Time to Failure.
   - Average Number of processor Failures.
   - The Mean Execution Time.
   - The Standard Deviation of the mean execution time.
   - Work.

3. The family of models includes, but is not limited to, the following cases:
   - case 0: (base case) Only busy processors can fail.
   - case 1: All processors (even when they are idle) can fail.
   - case 2: A fault-free processor exists, and idle processors cannot fail. That is, case 2 as applied to case 0.
   - case 3: A fault-free processor exists, and idle processors can fail. That is, case 3 as applied to case 1.
   - case 4: If the last processor fails before job completion, it is repaired (applied to either case 0 or case 1).
   - case 5: Processors that become idle are released, or decommissioned (as applied to any of the above).

4. We have analyzed results for the case were $N \gg P$ and $N = P$, examining in particular, the scaled behavior.

2 Background: Processors With No Failures

For definiteness, we define a job as a set of $N$ tasks, and let the mean time for a task to run in a uni-processor be $\tau$ seconds. Then the mean time it would take to run the job sequentially is simply $\tau N$. It is obvious that if all $N$ tasks take exactly the same time to execute, and are executed in parallel on $P$ processors, then the mean time (and the exact time) for all of them to be finished is

$$T(N, P|D) = \tau \left\lceil \frac{N}{P} \right\rceil.$$
The $D$ in $T(N, P|D)$ denotes that all the tasks take exactly the same amount of time, i.e., the distribution is Deterministic. We will call this ideal.

If the tasks are not all the same size and they are run in random order or, it is not known just how long a particular task will take until it is finished, then the mean time to finish all tasks is, in general, more complicated to derive, but there is a general formula when $P = N$. Following standard procedures from probability theory, let $\{X_i | 1 \leq i \leq N\}$ be the set of random variables (r.v.) denoting the time it takes to process each of the $N$ tasks. Assume they are independent and identically distributed. Then, in particular, $E(X_i) = \tau$ for all $i$. Let $F_X(x)$ be the Probability Distribution Function (PDF) for each of the task times, $R_X(x) = 1 - F_X(x)$ be the Reliability (or Complementary Distribution) function, namely,

$$R_X(x) \equiv Pr(X > x).$$

Also,

$$f_X(x) \equiv \frac{d}{dx} F_X(x)$$

is the probability density function (pdf). Let $Z_i(N)$ be the r.v. denoting the $i^{th}$ smallest task among the $X_i$’s. That is, the $Z_i(N)$’s are a reordering of the $X_i$’s such that $Z_1(N) \leq Z_2(N) \leq Z_3(N) \leq \ldots \leq Z_N(N)$. The study of these variables is called Order Statistics, see, e.g., [FELL71], [TRIV82]. The two r.v.’s relevant to this paper are $Z_1(N)$, and $Z_N(N)$. The latter variable corresponds to the time for the last task to finish, which is also the time that all tasks are finished. The probability that all $N$ tasks are finished by time $x$ is:

$$F_{Z_N(N)}(x) \equiv Pr(Z_N(N) \leq x) = [1 - R_X(x)]^N$$

and thus the mean time for all tasks to be finished is [using $\int_0^\infty xf(x) \, dx = \int_0^\infty R(x) \, dx$]

$$T(N, N|G) \equiv E(Z_N(N)) = \int_0^\infty R_{Z_N(N)}(x) \, dx = \int_0^\infty \left( 1 - [1 - R_X(x)]^N \right) \, dx.$$ 

The ‘G’ denotes any (General) distribution.

For $1 < P < N$, $T(N, P|G)$ may be very difficult to compute. However, for exponential distributions, the mean times can be calculated rather easily, because of their memoryless property. First note that in general, the reliability function for $Z_1(N)$ (the first task to finish) is the same as the probability that no tasks have finished by time $x$. That is,

$$R_{Z_1(N)} = [R_X(x)]^N.$$ 

For exponential distributions, $R_X(x) = e^{-x/\tau}$ so

$$R_{Z_1(N)} = e^{-Nx/\tau}.$$  

In other words, $Z_1(N)$ is also exponentially distributed, but with mean

$$E(Z_1(N)) = \frac{\tau}{N}.$$ 

6
Yes, the first of $N$ tasks running in parallel will finish in mean time $\tau/N$.

It is well known that the time remaining for an exponential process that has been running for some time is the same as though it just started (the memoryless property). This property is often used to derive properties such as the expectations for each of the $Z_i(N)$'s. After the first task finishes, there are $N-1$ tasks remaining, so the mean time for the second task to finish is given by:

$$E(Z_2(N)) = E(Z_1(N)) + \frac{\tau}{N-1}.$$  

In general,

$$E(Z_{i+1}(N)) = E(Z_i(N)) + \frac{\tau}{N-i}.$$  

It follows that (see, e.g [TRIV82, LIPS96]):

$$T(N, N|M) = E(Z_N(N)) = \tau \sum_{i=1}^{N} \frac{1}{i} = \tau H(N)$$

where $H(N)$ is the well-known harmonic series, and has the asymptotic property:

$$H(N) \rightarrow \log_e(N) + \gamma,$$

where $\gamma = 0.577215...$ is Euler's constant. (Note that by convention, $M$ is used to denote Memoryless, or Markovian - no-one seems to know which - when using exponential distributions.) In other words, the longest task is on the order of $\log(N)$ greater than the mean, and the job is not done until the last one is done.

Similarly, for $P$ processors, it can be shown that:

$$T(N, P|M) = \tau \left[ \frac{N}{P} + H(P) - 1 \right].$$  \hspace{1cm} (2)$$

Clearly, for $N \gg P$, time to completion approaches the ideal $\tau N/P$, with a constant additive term that becomes less significant with increasing $N$ (and constant $P$).

Keep in mind that all the above analysis is only valid for exponential task times. All other distributions (except for the deterministic) are more difficult to analyze. One might then question whether use of the exponential distribution is just a convenience, and has little to do with reality. But there are many reasons to support its use. One good reason applicable to this scenario is the following. It has been shown [FELL71] that if a job is broken up randomly into $N$ pieces, the lengths of the pieces will obey the formulas given above for exponential distributions.

The preceding discussion applies to, what we refer to here as, asynchronous task processing. That is, each processor executes a task independently of all other processors, and when finished, sends its results to the other processors, and promptly begins its next task, if there are any left. In many applications each processor, after finishing its task, must wait for all the others to finish theirs before going on to the next task. This is sometimes referred
to as a *fork-join construct*. We refer to it here as *synchronous task processing*. If all the tasks have the same processing time, then there is no difference in expected value between synchronous and asynchronous task processing, since they both are ideal (\(\tau[N/P]\)). But for exponential (and all other non-deterministic) distributions, there can be a significant synchronization cost. Since each job is executed in \(\lceil N/P \rceil\) cycles, the time to finish each is

\[
T_s(N, P) = \tau \left( \lceil N/P \rceil H(P) + H(P') \right),
\]

where \(\tau H(P)\) is the time for one cycle and \(P' := N - P \lceil N/P \rceil\) is the number of tasks left in the last cycle. The factor \(H(P)\) can be considered to be the *synchronization cost*. In comparing (2) and (3) we see that the difference between synchronous and ideal costs grows linearly with \(N\), whereas asynchronous costs are fixed relative to the ideal. For example, let \(P = 2\), and let \(N\) be even. Then \(H(2) = 3/2\), and

\[
T_s(N, 2) = \tau \frac{N}{2} = \tau \frac{N}{4}
\]

whereas

\[
T(N, 2) - \tau \frac{N}{2} = \frac{1}{2}.
\]

\(T_s\) can be reduced significantly by only synchronizing after each processor has performed exactly 2 tasks. Then, \(H(P)\) in (3) would be replaced by a smaller number. In our example, 3/2 would be replaced by 11/8, affecting a 12.5% decrease in time to processing a job. Even greater improvement can be gained by synchronizing after every 3 (or more) tasks.

In any case, for any non-deterministic distribution, for fixed \(P\),

\[
T_s(N, P) - \tau N/P = O(N),
\]

whereas,

\[
T(N, P) - \tau N/P = O(1).
\]

In the rest of this paper we will always be using asynchronous task processing, and assume that task times are exponentially distributed. The method can be extended (with difficulty) to non-exponential task times using linear algebraic methods (see, e.g., [LIPS92] or [NEUT81]).

3 Processors with Failures: The Base Model

In the previous discussions it was seen that parallel processing of tasks with unknown task times can lead to non-intuitive results. The introduction of the possibility that a task may not complete at all, yields results that appear to be counter-intuitive, because we now must deal with *conditional* probabilities, and conditional execution times. First we discuss another property from probability theory.
3.1 Mathematical Preliminaries

Suppose two processes, $X$ and $Y$, with distribution functions, $F_X(x)$ and $F_Y(x)$ begin at the same time. As described previously, the mean time for one of them to finish, is given by (recall, $Z_1(2) = \min(X, Y)$):

$$E(Z_1(2)) = \int_0^\infty R_X(x) R_Y(x) \, dx.$$  \hfill (4)

In general, $X$ will finish before $Y$ with probability:

$$Pr(X \leq Y) = \int_0^\infty f_X(x) R_Y(x) \, dx.$$  \hfill (5)

For instance let $Y$ be the random variable denoting the time that a processor fails, and $X$ is the random variable denoting the time that a task finishes. Then the probability that a task will finish before the processor fails ($q$) is:

$$q := Pr(X \leq Y) = \int_0^\infty f_X(x) e^{-\alpha x} \, dx.$$  \hfill (6)

If both $X$ and $Y$ are exponentially distributed, with rates $\alpha$ and $\lambda$, respectively, then

$$E(Z_1(2)) = \frac{1}{\alpha + \lambda}$$  \hfill (7)

and

$$q = Pr(X \leq Y) = \frac{\lambda}{\alpha + \lambda}.$$  \hfill (8)

These formulas are the basis of our model (and most continuous Markov chain models).

Devices that are subject to random uncorrelated failures are well represented by exponential inter-failure times. Now consider a single processor that is executing a task. From a modelling perspective, it is doing two independent things simultaneously. First it is processing the task at rate $\lambda := 1/\tau$, and secondly it is breaking down at rate $\alpha$, i.e., the mean time to failure is $1/\alpha$. Therefore, from (7), the mean time until something happens (either a failure, or a task completion) is $1/(\lambda + \alpha)$. This implies the following: the mean time for a task to complete, given that it does complete, is less than the mean time for completion in a fault-free environment. The reason for this is that longer tasks are more likely to fail before completion, so that finished tasks tend to be shorter.

From (8), the probability that the event (failure or completion) will be a task completion is given by:

$$q := \frac{\lambda}{\alpha + \lambda} = \frac{1}{1 + \gamma}, \quad \text{where} \quad \gamma := \alpha \tau.$$  \hfill (9)

The probability that the event is a processor failure is:

$$p := 1 - q = \frac{\gamma}{1 + \gamma}.$$  \hfill (10)
To give a physical meaning to \( \gamma \), consider (6). If \( 1/\alpha \ll \text{E}(X) = \tau \), then this can be approximated by:

\[
Pr(\text{task failure}) \approx 1 - \int_0^\infty (1 - \alpha x)f_X(x)\,dx = 1 - (1 - \alpha \tau) = \gamma.
\]

Equation (10) is a special case of this, and \( p \approx \gamma \) for \( \gamma \ll 1 \).

Next consider \( P \) processors, all doing the same things, simultaneously processing independent tasks, and risking failure. It follows from (1) that the mean time until an event occurs is simply \( 1/[(\lambda + \alpha)P] \), and the probability that the event will be a task completion is,

\[
p = \frac{\alpha P}{(\lambda + \alpha)P} = \frac{\alpha}{\lambda + \alpha} = \frac{\gamma}{1 + \gamma},
\]

the same as for one processor. This is most convenient. Extensions of the base model that we will present later do not share this property.

### 3.2 Base Model

The base model can be best explained by looking at Figure 1. Each node corresponds to a possible state, \((i, j)\), of the system. Index \( i \) labels the number of processors that have already failed, and \( j \) labels the number of tasks that have already finished. The reciprocal of the expression in each node is the mean time the system spends at that node if it gets there. Thus, for instance, node (1,1) corresponds to having 1 task completed and 1 processor already failed. The mean time until the next event is \( V(i, j) = 1/[(P - 1)\delta] \), where \( \delta = \alpha + \lambda \).

It may occur that at some point in the system’s evolution there could be more processors surviving than there are tasks remaining. In the base model, it is assumed that idle processors that are functional do not fail. In this case, the mean node time is \( 1/[(N - j)\delta] \), but \( p \) and \( q \) remain the same. In general, the rate of leaving each node is given by:

\[
B(i, j) = \delta \min(N - j, P - i),
\]

and the mean time spent there is \( V(i, j) = 1/B(i, j) \).

If the event at node \((i, j)\) is a task completion, then with probability \( q \), the state moves to the next node on the right. On the other hand, if the event is a processor failure, then the system moves with probability \( p \) to the next node below. If the system finally ends up at one of the smiling faces, then the job completed successfully, while if it ends at one of the frowning faces, the job failed. From this figure we can set up the recursive formulas for the various performance parameters.

#### 3.2.1 Probability of Job Failure

Define \( Pr(i, j) \) as the probability that the system will reach state \((i, j)\) during execution. Obviously,

\[
Pr(0, 0) = 1.
\]
Figure 1: Analytic Performance Model (Base Model) A node is identified by the indices $(i, j)$ where $i=$number of failed processors and $j=$number of tasks completed. The probability that something will happen at node $(i, j)$ in time $\Delta$ ($0 < \Delta \ll 1$) is given by $min(P-i, N-j)\delta \Delta$ where $\delta = \alpha + \lambda$. $\lambda$ is the service rate of a processor ($\tau = 1/\lambda$), and $\alpha$ is the failure rate of a processor when a task is running. The system progresses from state $(i, j)$ to state $(i, j + 1)$ by completing a task with probability $q$. A processor failure impedes the progress of the system with probability $p = 1 - q$ and causes a state-change from $(i, j)$ to $(i + 1, j)$. The job completes if it reaches a smiling face, and fails if it reaches a frown.

Along the top row and left-most column,

\[
Pr(0, j) = q Pr(0, j - 1) \quad \text{for} \quad 0 < j \leq N \\
Pr(i, 0) = p Pr(i - 1, 0) \quad \text{for} \quad 0 < i \leq P
\]

The other probabilities satisfy:

\[
Pr(i, j) = q Pr(i, j - 1) + p Pr(i - 1, j) \quad \text{for} \quad 0 < i < P, \quad \text{and} \quad 0 < j < N.
\]

It turns out that these equations are the same as those satisfied by the binomial distribution. Therefore,

\[
Pr(i, j) = \binom{i+j}{j} p^i q^j, \quad \text{for} \quad 0 \leq i < P, \quad \text{and} \quad 0 \leq j < N. \quad (12)
\]

The model extensions are not so fortunate.
Finally, we can calculate the probability that the job will finish with \( i < P \) failures, namely:

\[
Pr(i, N) = q Pr(i, N-1) = \binom{N+i-1}{i} p^i q^N, \quad \text{for} \quad 0 \leq i < P. \tag{13}
\]

For instance, the probability that the job will finish with no failures is simply \( Pr(0, N) = q^N \). The probability that the job will finish with fewer than \( P \) failures (probability of success) is \( S(N, P, \gamma) \) (abbreviated as \( S(N, P) \) when the meaning is clear) is

\[
S(N, P) = \sum_{i=0}^{P-1} Pr(i, N) = q^N \sum_{i=0}^{P-1} \binom{N+i-1}{i} p^i. \tag{14}
\]

The terms in the sum are the first \( P \) terms in the Taylor expansion for \( 1/(1-p)^N \), therefore it directly follows that \( \lim_{p \to 0} S(N, P) = 1 \).

It turns out that (13) and (14) are valid for any distribution, as long as \( p \) and \( q \) do not change from node to node. This is not to say that \( S(N, P) \) does not depend on the particular task time distribution. The connection occurs in evaluating \( q \) and \( p \) from (6). For instance, if the task times are all the same size then

\[
q = \int_0^\infty \delta(x-\tau) e^{-\alpha x} dx = e^{-\alpha \tau} = e^{-\gamma}
\]

(compare with value for exponential distribution, \( 1/(1+\gamma) \)), where \( \delta(x-\tau) \) is the Dirac delta function (see, e.g., [LIPS92]).

Next consider the probability that the job will fail when only \( j \) tasks have completed. This is given by,

\[
Pr(P, j) = p^j \binom{P+j-1}{j} q^j \tag{15}
\]

Obviously the probability of failure \( F(N, P) = 1 - S(N, P) \), and is also equal to:

\[
F(N, P) = \sum_{j=0}^{N-1} Pr(P, j). \tag{16}
\]

It follows from (15) and (16) that

\[
F(N, P) = O(p^P), \tag{17}
\]

showing clearly why \( F(N, P) \) is so close to 0 for small \( p \).

### 3.2.2 Periodicity Parameter

For this section only, we introduce the periodicity parameter, \( r \) [WEER01c]. To reduce communications costs, each processor could send its results after \( r \) tasks have been completed. This reduces the communications overhead, but if the processor fails, up to \( r \) tasks have to be redone. It can be shown that

\[
S(N, P, \gamma, r) = q^N \left[ \sum_{i=0}^{P-1} \binom{N+r + i - 1}{i} (1-q^r)^i \right]. \tag{18}
\]
Figure 2: Probability of Job Failure, \( F(N,P) \), (Base Model with Periodicity Parameter, \( r \)) Equation (18) is plotted for \( r = 1, 2, 4, 8, 16 \) in the range, \( 0 \leq \gamma < 0.007 \), with \( P = 8 \), and \( N = 1024 \). \( \gamma = \alpha \tau \) is a measure of the relative rates of failure and task completion. The graph demonstrates two significant properties. As expected, \( F(N,P) \) increases with increasing \( r \), but not appreciably. The probability that a job will fail can be ignored for small enough \( \gamma \), but above a certain point (in this case, above \( \gamma = .002 \)), the failure rate increases very rapidly to where it is unacceptable (in this case, \( \gamma > .005 \)).

Obviously, \( S(N,P,\gamma) = S(N,P,\gamma,1) \).

In Figure 2 \( F(N,P) \) is plotted as a function of \( \gamma \) for a typical system where \( N >> P \) for various values of \( r \). For this particular system, the probability of failure increases with \( r \), as expected, but the increase is not significant. Perhaps more significant is the rapid rise of \( F(N,P) \) above a certain value for \( \gamma \). In this case, for \( \gamma < .002 \) system failure will occur less than 0.1% of the time, but for \( \gamma > .005 \) system failure will occur over 15% of the time. The value of \( \gamma \) for which this rapid performance change occurs depends on the ratio \( P/N \). The larger the ratio, the higher is the changeover point.

### 3.2.3 Average Number of Processor Failures

The average number of processor failures, conditioned on a successful run, follows from (13) and is given by,

\[
\bar{n}(N,P) = \frac{\sum_{i=0}^{P-1} i Pr(i,N)}{\sum_{i=0}^{P-1} Pr(i,N)} = \frac{q^N \sum_{i=0}^{P-1} i \left( \begin{array}{c} N+i-1 \\ i \end{array} \right) p^i}{q^N \sum_{i=0}^{P-1} \left( \begin{array}{c} N+i-1 \\ i \end{array} \right) p^i} \tag{19}
\]
Since the results are conditioned on the job finishing successfully no more than \((P - 1)\) processor failures can occur. The limiting behavior of \(\tau(N, P)\) as \(\gamma\) increases, can be written in a closed form. First note that as \(\gamma\) increases, \(p = \gamma/(1 + \gamma) \to 1\). Therefore, (19) reduces to the following.

\[
\lim_{\gamma \to \infty} \tau(N, P, \gamma) = \frac{\sum_{i=0}^{P-1} i \binom{N+i-1}{i}}{\sum_{i=0}^{P-1} \binom{N+i-1}{i}}.
\]

By induction, it can be shown that,

\[
\sum_{i=0}^{P-1} \binom{N+i-1}{i} = \binom{N+P-1}{N}
\]

and,

\[
\sum_{i=0}^{P-1} i \binom{N+i-1}{i} = N \binom{N+P-1}{N+1}.
\]

Therefore, when \(\gamma \to \infty\) (or when \(p \to 1\)),

\[
\lim_{\gamma \to \infty} \tau(N, P, \gamma) = \frac{(P - 1)N}{(N + 1)} < (P - 1) \quad \forall \quad N.
\]

This tells us that no matter how bad things are \((p \to 1)\), if the job is to finish successfully, it will sometimes have more than 1 processor running at the end.

3.2.4 Average Number of Completed Tasks Before Failure

Here, for a moment, we look at the complementary process, the number of tasks that finish when a job fails. As in the previous section,

\[
\overline{\gamma}(N, P) = \frac{\sum_{j=0}^{N-1} j Pr(P, j)}{\sum_{j=0}^{N-1} Pr(P, j)}
\]

An interesting consequence of the formula is the following. Suppose that \(N \to \infty\). That is, there are an infinite number of tasks to be processed. Then, how many of them will be processed before the system fails? Let \(J\) be the r.v. denoting the number of tasks completed before failure. It can be shown that

\[
E(J) := \lim_{N \to \infty} \overline{\gamma}(N, P) = \frac{qP}{p}.
\]

This result is only valid for the Base Case.
3.2.5 Time for Job to Finish

We now turn our attention to the mean time it would take for a job to finish, given that it does not fail first. Before doing this we must define two functions:

\[ T(i, j) := \text{Mean time to reach and leave node } (i, j), \text{ assuming it is reached. This function cannot be computed directly. Instead we must first compute an auxiliary function, defined by:} \]

\[ T_p(i, j) := Pr(i, j) T(i, j). \]  \hspace{1cm} (21)

From their definitions, it is clear that:

\[ T(0, 0) = T_p(0, 0) = V(0, 0) = \frac{1}{P\delta}. \]

In a manner similar to the \( Pr(i, j) \)'s, along the top row and left-most column,

\[ T_p(0, j) = q T_p(0, j-1) + Pr(0, j) V(0, j) \quad \text{for } 0 < j \leq N \]
\[ T_p(i, 0) = p T_p(i-1, 0) + Pr(i, 0) V(i, 0) \quad \text{for } 0 < i \leq P \]

The other \( T_p \)'s satisfy:

\[ T_p(i, j) = q T_p(i, j-1) + p T_p(i-1, j) + Pr(i, j) V(i, j), \quad \text{for } 0 < i < P, \quad \text{and} \quad 0 < j < N. \]

Finally,

\[ T_p(i, N) = q T_p(i, N-1) \]  \hspace{1cm} (22)

and the mean time for the job to finish, with \( i < P \) processor failures is

\[ T(i, N) = \frac{T_p(i, N)}{Pr(i, N)}. \]  \hspace{1cm} (23)

Let \( T(N, P) \) be the (conditional) mean execution time given that the job does complete. Then

\[ T(N, P) = \frac{\sum_{i=0}^{P-1} T_p(i, N)}{S_N}. \]  \hspace{1cm} (24)

Although the variance of \( T(N, P) \) is not easy to compute, a good approximation can be made from the expression:

\[ \sigma^2(N, P) \approx \sigma^2_a(N, P) := \sum_{i=0}^{P-1} T(i, N)^2 Pr(i, N) - [T(N, P)]^2 \]  \hspace{1cm} (25)

Equations (24) and (25) will be discussed when illustrating results.
3.2.6 Mean Time To Failure

In an equivalent way to calculating the conditional mean time for a job to complete, we can calculate the conditional mean time to failure. Similar to (22) and (23), we can write,

\[ T_p(P, j) = p T_p(P - 1, j) \]  \hfill (26)

and the mean time for the system to fail given \( j \) tasks had finished is,

\[ T(P, j) = \frac{T_p(P, j)}{Pr(P, j)} \]

Thus,

\[ T_F(N, P) = \frac{\sum_{j=0}^{N-1} T_p(P, j)}{\mathcal{F}(N, P)} \]

and goes to MTTF(P) for \( N \to \infty \),

\[ MTTF(P) := \lim_{N \to \infty} T_F(N, P) \]

3.2.7 The Work Function

Of some interest to researchers in the field is \( \mathcal{W}(N, P) \), the cost, or processor-time product, or what we shall call here, the Expected Work function (or simply, work) [KANE97]. It may be defined as the total time spent by un-failed processors while a job is executing. This includes those that are idle (or duplicating tasks) when there are more processors available than there are tasks remaining. Work can be defined formally in the following way. Let \( K(t) \) be the r.v. denoting the number of processors that are still functional at time \( t \) during execution of a job. Then

\[ \mathcal{W}(N, P) := E \left( \int_0^{\text{done}} K(t) dt \right) \]

Obviously, if failures cannot happen \( (\alpha = 0) \), then (from (2)) \( \mathcal{W}(N, P) = P \cdot T(N, P) = P \cdot T(N, P|M) \). But if failures are possible another set of recursive equations is necessary. Recall that each node in Figure 1 represents time spent by the system. The number of processors that are still functional at that point is \( (P - i) \). This is sufficient to set up the equations.

As before, we define the function \( W_p(i, j) \) which, in analogy with \( T_p \), satisfies the following recursive equations:

\[ W_p(0, 0) = V(0, 0) P \]

and

\[ W_p(i, j) = q W_p(i, j - 1) + p W_p(i - 1, j) + Pr(i, j)V(i, j) (P - i) \]

Then

\[ \mathcal{W}(N, P) = \frac{\sum_{i=0}^{P-1} q W_p(i, N-1)}{S_N^P}. \]  \hfill (27)
3.2.8 Rerunning of Failed Tasks

In the introduction we discussed what happens when a task fails. In our model, we assume that it restarts from where it left off. Alternatively it is replaced by a new task from the same distribution that starts from the beginning (they both give the same answer). In many applications, the same task is restarted again from the beginning, and necessarily must run at least as long as it did before it failed previously. We now attempt to analyze the difference.

Consider the situation where a task is started over and over again until it finally runs to completion without failure. In our nomenclature this corresponds to $N = 1$ and $P = \infty$. In real systems without repair, $P$ must be finite, so there must be fewer than $P$ failures, either by a single task or several tasks. After that, there are no processors left, and the job as a whole fails. Therefore the analysis here is an upper bound to the time spent redoing tasks. Note, however, in systems with repair there could be an unbounded number of retries.

Let $t$ be the time that a particular task needs to run. Then $e^{-\alpha t}$ is the probability that the task will complete the first time it runs. The number of times it runs before success is geometrically distributed. Let $Pr_f(t)$ be the probability that a task of length $t$ will fail $i$ times, and succeed on the next trial. Then:

$$Pr_f(i|t) = (e^{-\alpha t})(1 - e^{-\alpha t})^i.$$ 

The expected number of failures before a success is given by:

$$\bar{\gamma}_f(t) := \sum_{i=0}^{\infty} i Pr_f(i|t) = e^{\alpha t} - 1.$$ 

The average time it takes each time it fails is given by the expression:

$$\bar{T}_f(t) = \int_0^t x e^{-\alpha x} dx / (1 - e^{-\alpha t}) = \frac{1}{\alpha} \left[ 1 - \frac{t}{e^{\alpha t} - 1} \right].$$ 

The mean time to finish the task is the number of failures, times the time per failure, plus the time for the final, successful run. In other words,

$$\bar{T}(t) = \bar{\gamma}_f(t) \bar{T}_f(t) + t = \frac{e^{\alpha t} - 1}{\alpha}.$$ 

It can be shown that $\lim_{\alpha \to 0} \bar{T}(t) = t$ (no failures), as it should. But note that for finite $\alpha$, the time to complete grows exponentially with increasing $t$.

Finally we evaluate the time it takes to finish a task, averaged over all possible task times $X$. That is,

$$\bar{\bar{T}} := \int_0^\infty \bar{T}(t)f_X(x) dx.$$ 

It should be clear that the integral for $\bar{\bar{T}}$ does not converge if $\lim_{t \to \infty} e^{\alpha t} f_X(t) \to \infty$. The class of heavy tail distributions [GRE199] fall in this category for all $\alpha > 0$. But so do...
exponential functions for which $\lambda \leq \alpha$. To get some idea of what the magnitude of $\tilde{T}$ could be, even for small $\alpha$, we replace $e^{\alpha t} - 1$ by $\alpha t + (\alpha t)^2/2$. Then

$$\tilde{T} \approx \int_0^\infty (t + \alpha t^2/2) f_X(t) dt = \tau + \alpha E(X^2)/2 = \tau \left[ 1 + \gamma \left( \frac{C^2+1}{2} \right) \right]$$

where $C^2 = \sigma^2/\tau^2$ is the coefficient of variation. This formula implies that for high variance distributions, systems without checkpointing with even a small failure rate could have problems!

Finally, we calculate $\tilde{T}$ for exponential distributions:

$$\tilde{T} = (1/\alpha) \int_0^\infty (e^{\alpha t} - 1) \lambda e^{-\lambda t} dt = \frac{\tau}{1 - \gamma}.$$ 

This is larger than our model would produce for $T(1, \infty)$ [See (24)]. For any $P$, it can be shown that:

$$T(1, P) \leq T(1, \infty) = \tau.$$ 

Clearly, if $\gamma$ is small, $P \ll N$, and task-time variance is not too large, our model is a good approximation, but if $\gamma \geq 1$ most systems are likely to be unstable if they do not have checkpointing.

### 4 Generalized Model

In this section we generalize the base model to accommodate several variations that may be of interest in terms of performance. In particular we consider the following five cases.

- **case 0**: Only busy processors can fail (Base Model).
- **case 1**: All processors can fail.
- **case 2**: A fault-free processor exists, and idle processors cannot fail. That is, **case 2** as applied to **case 0**.
- **case 3**: A fault-free processor exists, and idle processors can fail. That is, **case 3** as applied to **case 1**.
- **case 4**: If the last processor fails before job completion, it is repaired.

The algorithms for $S^M_N$, $T(N, P)$ and $W(N, P)$ are all the same as for the base case, but $p$ and $q$ now depend upon $(i,j)$. Also, $B(i,j)$ is more complicated. We now introduce some new definitions.
4.1 Formulas for the Generalized Model

Consider the following definitions:

Let $B_\alpha(i,j)$ be the rate at which processors are failing when at node $(i,j)$.

Let $B_\lambda(i,j)$ be the rate at which tasks are processed.

Let $p(i,j)$ be the probability that a failure occurred.

Let $q(i,j)$ be the probability of a task completion.

Then $B(i,j)$ from (11) becomes

$$B(i,j) = B_\alpha(i,j) + B_\lambda(i,j),$$

while $V(i,j) = 1/B(i,j)$ remains valid. Also,

$$p(i,j) = \frac{B_\alpha(i,j)}{B(i,j)} \quad \text{and} \quad q(i,j) = \frac{B_\lambda(i,j)}{B(i,j)}.$$

For the base model, $B_\alpha(i,j) = \alpha \min(N-j, P-i)$ and $B_\lambda(i,j) = \lambda \min(N-j, P-i)$, yielding the formulas given previously for $p$, $q$, and $B(i,j)$.

To determine the mean execution time (or other performance parameters) for cases 1 to 3, these new formulas are simply substituted in the expressions that were derived for the base model. The case 4 requires additional change. The Markov Chain Model for the generalized performance model is depicted in Figure 3. The difference between this model and the base model (Figure 1) is that in the base model $p$ and $q$ are the same at each node, whereas in the generalized model these probabilities (i.e. $p(i,j)$ and $q(i,j)$) are dependent upon the current state (or node) $(i,j)$.

4.2 Case 1: All Functioning Processors can fail

Under this scenario $B_\lambda(i,j)$ is the same as for the base case but

$$B_\alpha(i,j) = \alpha(P - i),$$

making the failure rate proportional to the number that have not failed as yet $[(P - i)]$, rather than the number that are active. For $N >> P$, there should not be much of a difference between this and the base model, since very few nodes have more processors available than tasks remaining. But for $N = P$, the difference might be significant for big enough values of $\gamma$.

4.3 Case 2: Failure-Free Processor, Idle Processors do not fail

Here too, $B_\lambda(i,j)$ is the same as in the base case. However, assuming that the fault-free processor is never idle,

$$B_\alpha(i,j) = \alpha \min(P-i-1, N-j).$$
Figure 3: **Generalized Analytic Performance Model** A node is identified by the indices \((i, j)\) where \(i=\)number of failed processors and, \(j=\)number of tasks completed. The probability that something will happen at node \((i, j)\) in time \(\Delta (0 < \Delta \ll 1)\) is given by \(B(i, j)\Delta\) where \(B(i, j) = B_\alpha(i, j) + B_\lambda(i, j)\). \(\lambda\) is the service rate of a processor \((\tau = 1/\lambda)\), and \(\alpha\) is the failure rate of a processor. The system progresses from state \((i, j)\) to state \((i, j+1)\) by completing a task with probability \(q(i, j)\). A processor failure impedes the progress of the system with probability \(p(i, j) = 1 - q(i, j)\) and causes a state-change from \((i, j)\) to \((i + 1, j)\). The job completes if it reaches a smiling face, and fails if it reaches a frown.

Note that, when only the fault-free processor is left \((i = P - 1)\), \(B_\alpha(P - 1, j) = 0\), which then implies that \(p(P - 1, j) = 0\), i.e., no more failures. Also, \(B(P - 1, j) = \lambda\), the rate of a single server.

Because there is no way to exit Figure 3 from the bottom, \(S^N_P = 1\) for all values of all parameters. That is, as expected, the job is guaranteed to complete, but for \(\gamma\) large enough, \(T(N, P)\) approaches \(\tau N\) (single processor time).

### 4.4 Case 3: Failure-Free Processor, Idle Processors Can Fail

Again, \(B_\lambda(i, j)\) is the same as in the base case. However, assuming that the fault-free processor is never idle,

\[
B_\alpha(i, j) = \alpha(P - i - 1).
\]

As in case 2, when only the fault-free processor is left \(p(P - 1, j) = 0\), and \(B(P - 1, j) = \)

20
\( \lambda \). Also, the job is guaranteed to complete, and \( T(N,P) \) approaches \( \tau N \) (single processor speed) for large \( \gamma \).

### 4.5 Case 4: Last Processor can be Repaired

Many other variants to the model can be employed. We mention two more here, although we shall not examine them further in this paper.

It is possible that all processors can fail, but when that happens, the last processor to fail can be repaired. Again we have 2 variants, depending on whether idle processors can fail. The \( B_\lambda \) and \( B_\alpha \) matrices are identical to cases 0 and 1, respectively, except for \( i = P - 1 \). At this point, the one remaining processor is repaired over and over again until the task completes. It can be shown that the mean time to complete a task here is \( \tau(1 + \alpha/\beta) \), where \( \beta \) is the (exponentially distributed) repair rate. Note that as \( \beta \to \infty \) (instantaneous repair) the mean time at node \((P - 1, j)\) becomes \( \tau \).

In summary, this pair of cases yields:

\[
V(P - 1, j) = \tau \left( 1 + \frac{\alpha}{\beta} \right)
\]

and,

\[
p(P - 1, j) = 0 \quad \text{and} \quad q(P - 1, j) = 1
\]

Otherwise, all parameters are the same as case 0 or 1, respectively. We mention that though the mean time for the task to finish when there is only one processor remaining is given by \( V(P - 1, j) \), it is not exponentially distributed.

### 4.6 Case 5: Other Possibilities

Another variant of the model would be to “decommission” the idle processors. That is, when the number of processors remaining exceeds the number of unfinished tasks, then the excess processors can be taken off-line. In this way they are not charged to the work total, but also, they are not available if one or more of the active processors should subsequently fail. This reduces the work function but increases the probability of total failure. Variants of this scheme can be studied, such as keeping 1 (or more) idle backups.

### 5 Some Calculations of the Base Model \((N \gg P)\)

Here we compare the performance parameters: probability that the system will fail \( \mathcal{F}(N, P) \), the average number of processor failures \( \mathcal{F}(N, P) \), the conditional mean execution time \( \mathcal{T}(N, P) \) and work \( \mathcal{W}(N, P) \) by varying the number of tasks \( N \) and the failure rate \( \alpha = \gamma/\tau \). The number of processors \( P = 8 \) and the mean execution time \( \tau = 1.0 \) unit. A summary of the overall behavior and how results are scaled is discussed in this section. Note that for explanation purposes, \( \mathcal{F}(N, P) \) is sometimes written as \( \mathcal{F}(N, P, \gamma) \). The same is true for other performance parameters.
Figure 4: Probability that the System will Fail as a function of $\gamma$ and of $\gamma \times N$: The probability that the system will fail (left graph) increases with $\gamma$ and with $N$. The failure probability of the system (right graph) scales well when the failure rate $\alpha = \gamma/\tau$ is increased by a factor of $N$.

5.1 Failure Probability

$F(N, P)$ (see Figure 4 left graph) increases with $\gamma$ and with $N$. The more tasks to be processed, the higher the probability of processor failures, and hence to a higher failure probability of the system. $F(N, P)$ (right graph) scales well when the failure rate $\alpha = \gamma/\tau$ is increased by a factor of $N$. A point of inflection occurs, in all cases, when $\gamma \times N = \alpha \times \tau \times N \approx 7.5$. This point corresponds to a system failure probability of 50%. A close approximation (as demonstrated in the right graph) of the $F(N, P, \gamma)$ for different values of $N$ can be obtained from the following,

$$F(N_1, P, \gamma/N_1) \approx F(N_2, P, \gamma/N_2). \quad (28)$$

5.2 Mean Number of Failed Processors

We next show some calculations of (19). One would expect $\bar{\tau}(N, P)$ to approach $P - 1$ with increasing $\gamma$, as seems to be the case in the left graph of Figure 5. But in Section 3.2.3 we showed that the asymptotic value is $(P - 1) N/(N + 1)$. Taking this into account, we scaled $\bar{\tau}(N, P)$ accordingly. The results are plotted in the right graph of Figure 5. The curves are almost identical for $N >> P$, validating the scaling law:

$$\bar{\tau}(N_1, P, \gamma/N_1) \frac{N_1 + 1}{N_1} \approx \bar{\tau}(N_2, P, \gamma/N_2) \frac{N_2 + 1}{N_2}. \quad (29)$$

The closeness of the $\bar{\tau}(N, P, \gamma)$’s is shown in Figure 6, where the difference between $\bar{\tau}(2N, P, \gamma*2N)$ and $\bar{\tau}(N, P, \gamma*N)$ is plotted. The differences are always less than $6.5 \times 10^{-3}$,
Figure 5: **Average Number of Failed Processors**, as a function of $\gamma$ and of $\gamma \times N$: The average number of processor failures (left graph) increases with $\gamma$ and with $N$. Note that, the results are conditioned on the job finishing successfully and hence no more than $(P-1)$ processor failures can occur. The average number of failures (right graph) scales well using the rule (29).

Figure 6: **Convergence of $\tau(N, P)$ as a function of $\gamma \times N$**: This graph further justifies the scaling of $\tau(N, P)$ as achieved by (29) for increasing values of $N$. At $\gamma \times N = 6$, the curves appear to be intersecting. However, they intersect at slightly different points.

and get smaller linearly with increasing $N$. The apparent multiple intersection near $\gamma N = 6$ is not exact.

### 5.3 Time of Job Execution

$T(N, P, \gamma)$ (see (24)), the conditional mean execution time (Figure 7 left graph) increases for increasing values of $\gamma$ and $N$. For small $\gamma$, the slope of $T(N, P, \gamma)$ actually increases as well. But finally, a point of inflection occurs due to the fact that jobs that would have taken
Figure 7: **Conditional Mean Execution Time and Ratio as a function of $\gamma$ and of $\gamma \times N$:**
The conditional mean execution time (left graph) increases for the given values of $\gamma$ and $N$. The conditional mean execution time ratio (right graph) scales well only for small values of $\gamma$.

longer are more likely to fail and thus are less likely to be included in the average. Since, $\mathcal{F}(N, P, \gamma)$ also increases with $N$, the point of inflection is more apparent for large values of $N$. All the curves appear to be approaching a horizontal asymptote, but this is not true. For large enough $\gamma$, their slopes become negative. For those values of $\gamma$, $S(N, P, \gamma)$ is extremely small ($<10^{-6}$), and the only chance a job has to finish is if it is “lucky” enough to have extremely small task times.

The ratio, $\mathcal{T}(N, P, \gamma)/\mathcal{T}(N, P, 0)$ (Figure 7 right graph) scales well only for small values of $\gamma$. The different curves peak at different places, and we were not able to find an appropriate scaling factor to align them better.

### 5.4 Work

The amount of work done, $\mathcal{W}(N, P, \gamma)$ (Figure 8 left graph) by the processors increases with $N$ and decreases with $\gamma$. Of course, the more tasks to be processed the more work is to be done. However, the work ratio (Figure 8 right graph) shows clearly that $\mathcal{W}(N, P, \gamma)$ decreases with increasing $\gamma$. This is caused by two mechanisms. First, there is a decrease in the total idle-time of processors since more and more processors fail. The idle-time at $\gamma = 0$ is given by $PH(P) - P$. For $\gamma > 0$, the number of functional processors, and thus the number of idle processors and total idle time will be less. Second, longer jobs tend to fail more often than shorter ones, so there is less work to be done for those jobs that finish.

### 6 Results for the Generalized Model

In this section we present two sets of performance results. The first set corresponds to the general scenario in which $N >> P$. The second set corresponds to the (massively parallel)
Figure 8: Work and Ratio as a function of $\gamma$ and of $\gamma \times N$: The amount of work done by the processors increases with $N$ and decreases with $\gamma$. The work ratio (right graph) shows clearly that $W$ decreases with increasing $\gamma$.

scenario where $N = P$. Each graph consists of four curves that describe the behavior for each of the four cases (listed in Section 4).

Figure 9: Probability that System will Fail and Difference, as a function of $\gamma$: The probability that the system will fail for cases 0 and 1 (left graph) increases as $\gamma$ increases. For cases 2 and 3 it is 0 since there is one fault-free processor. The failure probability difference (right graph) for cases 0 and 1 is always small ($<10^{-5}$) and peaks at around $\gamma = 0.0075$. This point corresponds to a failure probability of 0.5 (i.e. 50% of the time, the system fails). When $\gamma > 0.0075$, the difference diminishes.
6.1 Results for the General Case for $N >> P$

The probability that the system will fail (Figures 9 left graph) increases rapidly near $\gamma = 1.0$ ($p = 1/2$) for cases 0 and 1. However the probability is slightly higher for case 1 than for case 0 since idle processors can also fail in case 1. The probability of system failure for cases 2 and 3 is 0 because there is one fault-free processor. The system failure probability difference (Figures 9 right graph) for cases 0 and 1 is always small ($< 10^{-5}$) and peaks at around $\gamma = 0.0075$ This point corresponds to a failure probability of 0.5 (i.e 50% of the time, the system fails). When $\gamma > 0.0075$, the difference diminishes.

![Graphs showing Mean Execution Time and Difference as a function of $\gamma$]

Figure 10: **Conditional Mean Execution Time and Difference, as a function of $\gamma$**: For all cases the mean execution time (left graph) increases as $\gamma$ increases. However, when $\gamma > 0.003$, the slope of cases 0 and 1 approaches 0 as $\gamma$ increases. The mean execution time for cases 2 and 3 approaches the execution time of a single processor since there is one fault-free processor. The mean execution time difference (right graph) for cases 0 and 1 is positive for small values of $\gamma \leq 0.0008$, since more processors are likely to fail in case 1 than in case 0. The time difference (too small to show on the graph) for cases 2 and 3 is positive because in case 2, idle processors do not fail.

For all cases (Figures 10 left graph) the mean execution time increases as $\gamma$ increases. However, when $\gamma > 0.003$, the slope of cases 0 and 1 approaches 0 as $\gamma$ increases. This is because the mean execution time is conditioned on completion of a job. The average time for these jobs tend to be shorter because the jobs that would have taken longer failed and thus were not included in the average. The mean execution time for cases 2 and 3 approaches the execution time of a single processor since there is one fault-free processor. The mean execution time difference (Figures 10 right graph) for cases 0 and 1 is positive for small values of $\gamma \leq 0.0008$ since more processors are likely to fail in case 1 than in case 0. However, when $0.0008 < \gamma < 0.0075$, the mean execution time difference becomes negative since in case 1, all processors can fail and thus the issue on the execution time being conditioned begins to dominate. Then again for $\gamma \geq 0.0075$ the difference is positive since the former issue where in case 1 more processors are likely to fail than in case 0.
begins to dominate. The time difference (too small to show on the graph) for cases 2 and 3 is positive because in case 2, idle processors do not fail and hence it takes less time to complete a job than with case 3. Here the conditional time is not an issue since one processor is fault-free.

![Graph showing Work Performance with N=1024 P=8 Mean Task Time](image)

Figure 11: **Work and the Average Number of Failures, as a function of γ.** The amount of work done (left graph) for cases 0 and 1 decreases as γ increases. In cases 2 and 3, the work done also decreases but eventually reaches 64 time units which is the work of the single fault-free processor. The average number of failures (right graph) for cases 0 and 1 have a higher slope than the other two cases since there is no fault-free processor. However, for higher values of γ, a crossover point occurs at around γ = 0.0075.

The amount of work done (Figures 11 left graph) for cases 0 and 1 decreases as γ increases. In cases 2 and 3, the work done also decreases but eventually reaches 64 time units which is the work of the single fault-free processor. In all four cases the work decreases since for exponential task-times, the product of the active processors and the harmonic sum decrease with failures even though tasks have to be redone. Note that this is opposite to what occurs to work-done for deterministic task-times.

The average number of failures (Figures 11 right graph) for cases 0 and 1 have a higher slope than the other two cases since there is no fault-free processor. However, for higher values of γ, a crossover point occurs at around γ = 0.0075. This point corresponds to a system failure probability of 0.5. In all cases, the average number of failures appears to approach $P - 1$ (i.e., 7 in this example) as γ increases, but as given in (20), we know that the asymptotic value for case 0 is $7 \times 1024/1025 = 6.99317$.

### 6.2 Results for $N = P$

In the previous section we choose $P = 8$, thus at most 7 tasks out of 1028 ($< 1\%$) tasks can fail, so there is very little difference between cases 0 and 1 (and between 2 and 3). But now we set $N = P = 128$, so the differences between the cases can be significant.
Figure 12: Probability that System will Fail and Difference as a function of $\gamma$ The probability that the system will fail (left graph) increases rapidly near $\gamma = 1.0$ ($p = 1/2$) for cases 0 and 1. However the probability is higher for case 1 than for case 0 since idle processors can also fail in case 1. The probability of failure for cases 3 and 4 is 0 because there is one fault-free processor. The failure probability difference (right graph) for cases 0 and 1 peaks at around $\gamma = 0.9$. This point corresponds to a failure probability of 0.5 (i.e., 50% of the time, the system fails). When $\gamma > 0.9$, the difference diminishes.

As expected the probability that the system will fail (Figure 12 left graph) increases rapidly near $\gamma = 1$ ($p = 1/2$) for cases 0 and 1. However the probability is higher for case 1 than for case 0 since idle processors can also fail in case 1. The probability of system failure for cases 3 and 4 is 0 because there is one fault-free processor. Even though cases 0 and 1 are not fault-free, the probability of system failure is negligible until $\gamma$ approaches 1. After all, from (17), $\mathcal{F}(P, P) = O(q^p)$. It can be shown that this curve approaches a step function at $\gamma = 1.0$ as $N = P \to \infty$. The system failure probability difference (Figure 12 right graph) for cases 0 and 1 peaks at around $\gamma = 0.9$. This point corresponds to a system failure probability of 0.5. When $\gamma > 0.9$, the difference diminishes.

The mean execution time (Figure 13 left graph) for cases 0 and 1 increases slightly and then decreases as $\gamma$ increases. The initial rise is due to the probability that at some point during execution there can be more tasks remaining than there are processors remaining. As $\gamma$ increases this probability also increases, but then at some point job failure becomes significant, and longer jobs are conditioned out of the average, so the mean time for those jobs that finish decreases. For cases 2 and 3, the execution time approaches the execution time on a single processor because one processor is fault-free. The time difference (Figure 13 right graph) for cases 0 and 1 is negative because in case 1, all processors can fail and thus the issue on the execution time being conditioned applies here. However, time difference for cases 2 and 3 is positive because in case 2, idle processors do not fail so there are more processors available. Hence it takes less time to complete a job than with case 3. Here the conditional time is not an issue since one processor is fault-free and all jobs finish successfully.

28
Figure 13: Conditional Mean Execution Time and Difference as a function of $\gamma$ The conditional mean execution time (left graph) for cases 0 and 1 increases and then decreases as $\gamma$ increases. This is because the execution time is conditioned on completion of a job. For cases 2 and 3, the execution time approaches the execution time on a single processor because one processor is fault-free. The time difference (right graph) for cases 0 and 1 is negative because in case 1, all processors can fail. However, time difference for cases 2 and 3 is positive because in case 2, idle processors do not fail.

Figure 14: Work and Average Number of Processor Failures, as a function of $\gamma$ In all cases, the amount of work done (left graph) decreases as the failure rate increases. In cases 2 and 3, the work done reaches 128 $\tau$ units, the work of the fault-free processor. The average number of processor failures (right graph) for cases 1 and 3 have a higher slope than the other two cases since, idle processors can also fail.

In all cases (Figure 14 left graph), the amount of work done decreases as the failure rate increases. This model assumes restart at the point where failure occurred. That is, no work is lost. Furthermore, when processors fail, there are fewer idle processors (whose time is charged to work) waiting for the job to complete. In cases 2 and 3, the work done
reaches 128 $\tau$ units, the work of the fault-free processor. For cases 0 and 1, at some point the work actually drops below $P \cdot \tau$, showing clearly that only shorter jobs are finishing successfully.

Observe (Figure 14, left) that when $\gamma$ is small jobs don’t fail, so cases 0 and 2 have almost identical work, while cases 1 and 3 are almost identical. But as $\gamma$ increases 0 and 1 converge on each other, and 2 and 3 converge. For small $\gamma$, the same is true for the average number of failed processors (Figure 14 right graph), but for large $\gamma$, case 0 approaches the asymptotic value given by (20), namely $(P - 1)N/(N + 1) = 127 \cdot 128/129 = 126.0155$. The other 3 cases appear to approach $P - 1 = 127$, as expected. Finally, we note that the average number of failures for cases 1 and 3 have a higher slope than the other two cases since idle processors can also fail.

7 Conclusion

We have analyzed a number of different performance parameters of a system that performs $N$ independent tasks using $P$ fault-prone processors. In the base model, curves were analyzed for scaled behavior and close approximations were found for the failure probability of the system and the average number of processor failures for different values of $N$ and $\gamma$. However the conditional mean execution time and the work done for different values of $N$ scaled well only for small values of $\gamma$.

In the generalized model we focused on two systems with the following characteristics. In the first system (call it $A1$) all processors were fault-prone (i.e. jobs can fail) while in the second system (call it $A2$) we assumed that there is one fault-free processor (i.e. jobs can never fail). The system $A1$ was analyzed for the two cases where in the first case (case 0) idle processors where not allowed to fail while in the second case (case 1) idle processors were allowed to fail. Then $A2$ was also analyzed for the above two cases (we called it case 2 and case 3). The results between case 0 and case 1 were similar. The same was true for case 2 and case 3. However, there were significant differences between cases 0 and 2 and between cases 1 and 3.
References


