computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

AVAILABILITY VERSUS PERFORMANCE

Pierre M. Fiorini [1], Lester Lipsky [2]

[1] Department of Computer Science, University of Southern Maine, Portland, ME, USA, pfiorini@usm.maine.edu, www.cs.usm.maine.edu
[2] Department of Computer Science & Engineering, University of Connecticut, Storrs, CT, USA, lester@engr.uconn.edu, www.cse.uconn.edu

Abstract: *We discuss analytic procedures for evaluating the availability of parallel computer systems comprised of P processors with N tasks subject to failures and repairs. In addition, we argue, via analytic and numeric examples, that not incorporating the task-stream into the model is an inadequate approach for evaluating system performance.*

Keywords: – *Parallel and Distributed Processing, Performance Evaluation, Performability, Queueing Theory.*

1. INTRODUCTION

There has been much research in the area of studying the *availability* of parallel processing systems **(**PPS**)** subject to failure and repair [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]. In particular, they consider a system of *P* independent processors with

$$\left[p_a(x)\right]_k = \Pr\left[K(x) = k\right] \text{ for } 0 \le k \le P \quad (1)$$

where $K(x)$ is the random variable (r.v.) denoting the number of processors that are functional **(**available**)** at time *x*. This, and other performance parameters such at *Mean Time To First Failure* **(**MTFF**)** and *Mean Time Between Failures* **(**MTBF**)**, have been studied in detail [9]. Some researchers have even been able to solve for systems where failure and repair times have general probability distributions.

This is an important contribution to ascertaining the reliability of any system. However, this approach is inadequate for evaluating the productivity, or performance, of a system as to processing a set of *N* tasks (hereafter referred to as the *task-stream*) since it is (often implicitly) assumed that all resources that are available at any time will somehow be used. The implication of this is that system availability is *independent* or *decoupled*) from the workload running on the system.

Techniques have been developed that incorporate the dependability and performance aspects of an unreliable computing system in a performance model. For instance, *Markov Reward Models* (MRM's) are commonly used to assess system *performabilty* [2] [12]. By definition, performability models characterize the interaction between the availability of a computing system and its performance [4]. Often times, when using MRM's, researchers and practioners *decouple* the dependability and performance aspects of the model. For instance, separate models are created that represent system availability and performance respectively. Each of the models are solved separately and later combined to generate system performance measures (for some examples see [12]). The reasoning behind this technique is: 1) When the problem is formulated in this manner, it is easier to solve numerically; and 2) This approach can significantly reduce the required state space [12]. Unfortunately, if availability and performance are separated in this way, then results can be inadequate since most computing systems are too dynamic to be represented in this manner. The reason is that in an unreliable system, tasks can potentially see one or more changes in the number of active processors during their lifetime.

Thus, in order to properly describe the execution of tasks in a changing environment, execution, failure, and repair must be treated together. One way to do this is utilize MRM's that do not decouple system dependability and performance. These types of models are known as *integrated performability models*. In general, these models are more desirable than MRM's that separate system availability and performance because the task-stream is more faithfully represented. In other words, there will always be a time when there is not enough work

available to keep all available processors busy and these models capture that behavior. Indeed, in some very interesting cases, the interaction between execution and failure/repair is explicit. For instance, if processors are subject to failure only when they are actively processing tasks, then availability cannot be decoupled from the workload, for there are times when there are fewer tasks in the systems than there are available processors.

2. CONTRIBUTIONS

In this paper, we illustrate how a matrix analytic approach can be to calculate expected performance measures (e.g., availability) for an unreliable computing system given any number of tasks and processors, assuming the failure, repair, and task times are exponentially distributed. A complete analysis for non-exponential distributions is reserved for future research, however, much insight can be gained from this model.

In addition, we investigate how availability affects, and is affected by the task stream. We do this by analyzing the job via *epochs* or *task completion points* from which much insight can be gained. Comparable work in this area has been done using PH distributions, however, these approaches consider the distribution of the *entire* job and not individual task completion points (see, for example, [13]).

Furthermore, it can be shown by using our technique, the state space required to represent the process of *execution - failure – repair - execution* is substantially reduced. For example, the method proposed by [13] requires $N \cdot (P + 1)$ states to represent this process. The reason is that *Kronecker products* are used (see [13]). On the other hand, our method requires $(P + 1)$ states – an improvement is by a factor of N. This is important consideration for whatever algorithm is utilized to compute performance measures - especially as N and P get large.

3. THE MODEL

Consider a system with P identical, independent processors that can fail, at exponential rate α. When failed, they are repaired (one or more at a time, depending on how many repairmen there are - here assumed to be 1) at exponential rate β. A job, made up of N independent and identically distributed (iid) tasks, must run on this system. Up to P tasks run simultaneously, the rest reside in a waiting line. If a processor fails while executing a task, the task goes back in the waiting line, but when it restarts later, it continues where it previously left off. It is assumed task times are exponentially distributed, with mean, $\tau = 1/\lambda$. A state, (k, j), in our model represents the number of processors that are up ($0 \le k \le P$), an the number of tasks that have completed ($1 \le j \le N$). If one of the simultaneously executing tasks finishes when the system is in state (k, j), then the j^{th} task has completed. The state of the system moves up and down as processors fail and are repaired, and moves in a feed-forward manner to the right when one of the active tasks finishes. The period between departures is called an *epoch*, which we denote by j. When an epoch completes, these are called *embedding points* or *epoch completion points* and indicates that a task has completed. For example, when the first epoch completes, then the first task finishes. Thus, the system enters state $(k, j + 1)$. The job is completes when the system transitions from one of the states when $j = N$.

Let

$$y(k, j) = \min[k, N - j + 1] \qquad (2)$$

be the number of active tasks when there are k functional processors in the j^{th} epoch. Then the mean time spent at state (k, j) is $1/\mu(k, j)$ for

$$\mu_\alpha(k, j) = \mu_\alpha(k, j) + \mu_\lambda(k, j) + \mu_\beta(k, j).$$

Two different models for the rate at which processors fail are:

$$\mu_\alpha(k, j) = \begin{cases} \alpha k & \text{if idle processors fail;} \\ \alpha y(k, j) & \text{if active processors fail.} \end{cases}$$

Also,

$$\mu_\lambda(k, j) = \lambda y(k, j)$$

is the rate at which tasks finish, and

$$\mu_\beta(k, j) = \begin{cases} 0 & k = P \quad \text{(no repairs necessary);} \\ \beta & k < P \quad \text{(one repaired at a time).} \end{cases}$$

is the rate at which processors are repaired. Obviously, other choices for μ_β are possible (e.g., $\mu_\beta(k, j) = \beta(P - k)$ – all processors have a self-repair capability).

Define the k^{th} component of the *state vector* (a row vector), $[\mathbf{p}(j)]_k$, to be the probability that there are k functional processors at the start of the j^{th} epoch. Then $\mathbf{p}(j) \cdot \varepsilon' = 1$ for all j, where ε' is the column vector with all 1's. In a normal state of operation, a job would begin with all processors functional. In this case the job starts in state $(P, 1)$ (the beginning of the first epoch), so

$$\mathbf{p}(1) = [1, 0, 0, ..., 0],$$

Note that for convenience we place the elements in reverse order. That is,

$$\mathbf{p}(j) = [p_P(j), p_{P-1}(j), p_{P-2}(j), ..., 0].$$

There could be a scenario where if the system fails (the system finds itself in state (0, *j*) then the job must start from the beginning. In that case it could restart when the first processor is repaired (state (1, 1)). But now

$$\mathbf{p}(1) = [0, 0, ..., 1, 0].$$

If one considers the set of states in column *j* to describe the j^{th} epoch, then the $(P + 1) \times (P + 1)$ *infinitesimal generator matrix* is given by:

$$\mathbf{B}(l) = \begin{bmatrix} \mu(P, j) & -\mu_\alpha(P, j) & ... & 0 \\ -\mu_\beta(P, j) & \mu(P-1, j) & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & -\mu_\alpha(1, j) \\ 0 & 0 & ... & \mu_\beta(0, j) \end{bmatrix} \quad (3)$$

where $l = N - j + 1$ is the number of tasks remaining. Again, the columns are listed in reverse order. For instance, $[\mathbf{B}(l)]_{PP} = \mu(P, j)$, and $[\mathbf{B}(l)]_{00} = \mu_\beta(0, j)$. The reason for using *l* instead of *j* for labelling the **B**'s is that $\mathbf{B}(l_1) = \mathbf{B}(l_2)$ for all $l_1, l_2 \geq P$, independent of *N*, if $N > P$.

The vector-matrix pair, $<\mathbf{p}(j), \mathbf{B}(l)>$ generates the evolution of the system during any epoch, for it can be shown that the *Reliability Matrix*, defined by:

$$\mathbf{R}(j|t)_{ik} := [\exp(-B(l)t)]_{ik} = \Pr(K_j(t) = k \mid i)$$

has the following meaning: Given that the j^{th} epoch started with *i* functional processors, at time *t* there will be *k* functional processors, and no task will have finished. Therefore, $[\mathbf{R}(j)\varepsilon']_i$ is the probability that the j^{th} epoch will end after time *t*, given that it started with *i* functional processors.

Note that in (1) *x* refers to the time since the *job* began, while here, *t* is the time since the j^{th} epoch began. Thus $K_j(t)$ is the r.v. denoting the number of functional processors at time *t* since the j^{th} epoch began. The epoch points and *x* are not directly related. Given some time *x* one would have to find the probability that *j* tasks have finished (or that the system is in epoch $(j + 1)$). On the other hand, the j^{th} epoch could have begun at any time, so one would have to find the probability that the j^{th} epoch began at time *x*.

From the definition of $\mathbf{p}(j)$, it follows that

$$\mathbf{R}(j \mid t) = \mathbf{p}(j)\mathbf{R}(j \mid t)\varepsilon'$$

is the reliability function for the j^{th} epoch. Let $\mathbf{V}(l) := [\mathbf{B}(l)]^{-1}$, then $[\mathbf{V}(l)]_{ik}$ is the mean time that there are *k* functional processors during the j^{th} epoch, given that the epoch started with *i* functional processors. Finally, we have the mean time for the j^{th} epoch:

$$T(j) = \mathbf{p}(j)\mathbf{V}(l)\varepsilon' \quad (4)$$

The total time to complete the job is then

$$\mathrm{T}(N) = \sum_{j=1}^{N} T(j) \quad (5)$$

We next define the *Completion rate matrix* for the j^{th} epoch:

$$\mathbf{\Lambda}(l) = \lambda \begin{bmatrix} y(P, j) & 0 & 0 & ... & 0 & 0 \\ 0 & y(P-1, j) & 0 & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & y(1, j) & 0 \\ 0 & 0 & 0 & ... & 0 & 0 \end{bmatrix} \quad (6)$$

and follows that $\mathbf{B}(l)\varepsilon' = \mathbf{\Lambda}(l)\varepsilon'$. Therefore, it can be shown the matrix, $\mathbf{Y}(l)$, defined by

$$\mathbf{Y}(l) = \mathbf{V}(l)\mathbf{\Lambda}(l)$$

satisfies $\mathbf{Y}(l)\varepsilon' = \varepsilon'$, i.e., it is a Markov matrix. It's meaning is as follows. Given that epoch *j* started with *i* functional servers, $[\mathbf{Y}(l)]_{ik}$ is the probability that the next epoch will start with *k* functional processors. Therefore, we can write:

$$\mathbf{p}(j+1) = \mathbf{p}(j)\mathbf{Y}(l) \quad (7)$$

As before, $l = N - j + 1$, and all $\mathbf{V}(l)$, $\mathbf{\Lambda}(l)$, and $\mathbf{Y}(l)$ are independent of *l* for $l \geq P$. What we have described above **(**the sequence of times, $T(j)$), is known as a *Markov Renewal Process*. Only if $\mathbf{Y}(l)$ is of rank 1, and is independent of *l* does the process become a simple (actually, delayed) renewal process. For then each epoch would be independent

of the previous one, and the $T(j)$'s, except for $T(1)$, would be iid.

4. AVAILABILITY IN THIS MODEL

We now examine where availability fits into our model. If we define *Availability* as the total time available on functional processors, then this can be calculated by defining the *Processor Availability Matrix*,

$$\mathbf{A}(l) = \begin{bmatrix} P & 0 & 0 & ... & 0 & 0 \\ 0 & P-1 & 0 & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & 1 & 0 \\ 0 & 0 & 0 & ... & 0 & 0 \end{bmatrix}$$

Then

$$A(j) = \mathbf{p}(j)\mathbf{V}(l)\mathbf{A}\varepsilon' \qquad (8)$$

is the total time that processors are available during the j^{th} epoch. But

$$\lambda A = \mathbf{\Lambda}(l) \quad \forall\, l \le N - P,$$

therefore (where $\tau = 1/\lambda$),

$$A(j) = \mathbf{p}(j)\mathbf{V}(l)\tau\mathbf{A}\varepsilon' = \tau\mathbf{p}(j)\mathbf{Y}(l)\varepsilon' = \tau,$$

since $\mathbf{Y}(l) = \mathbf{V}(l)\mathbf{\Lambda}(l)$ and $\mathbf{Y}(l)\varepsilon' = \varepsilon'$ for all l.

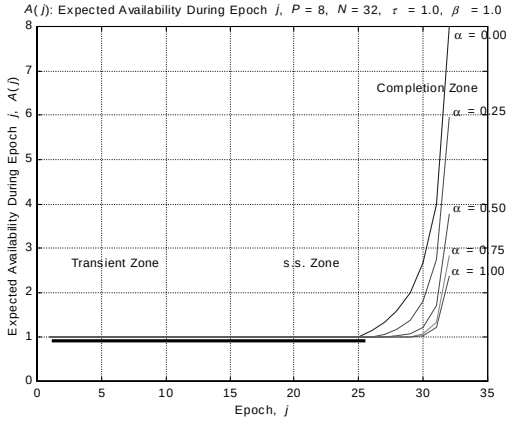


Fig 1 - The expected time processors are available during the j^{th} epoch, $A(j)$, when P = 8, N = 32 β = 1.0, and τ = 1.0 for α between 0.00 to 1.00 inclusive.

Another parameter, *processor activity*, becomes trivial in this model where the assumption is that all tasks restart where they left off. Since $v(j)_k = [\mathbf{p}(j)\mathbf{V}(l)]_k$ is the total time there are k functional processors during epoch j, then using (2), $\sum_k v(j)_k y(k, j)$ is the total time spent processing tasks in that epoch. But from (6), $y(k, j) = \tau[\mathbf{\Lambda}(l)]_{kk}$, so

$$\sum_{k=1}^{P} v(j)_k y(k, j) = \tau\mathbf{p}(j)\mathbf{V}(l)\mathbf{A}\varepsilon' = \tau\mathbf{p}(j)\mathbf{Y}(l)\varepsilon' = \tau.$$

In other words, the useful activity in each epoch is exactly the time needed to process one task. Thus the total useful activity to finish the whole job is τN. As long as $j \le N - P$ there are always enough tasks to keep the processors busy, so for those epochs, *availability* = *activity*. Fig. 1 demonstrates this behavior.

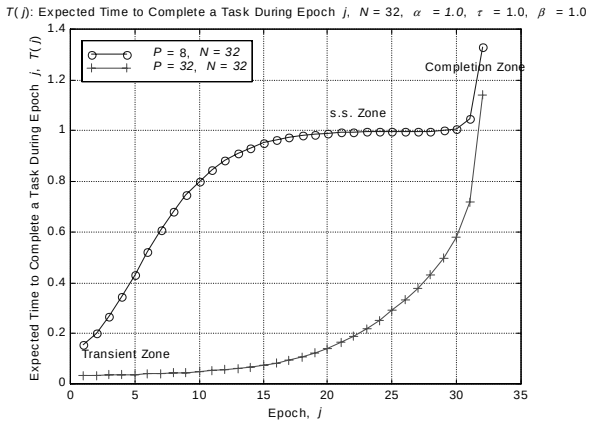


Fig 2 - The expected time time to complete the j^{th} epoch, $T(j)$, when P = 8, 32 and N = 32 when α = 0.50, β = 1.0, and τ = 1.0.

Next observe that $\mathbf{B}(l)$ and $\mathbf{\Lambda}(l)$, and thus $\mathbf{V}(l)$, $\mathbf{Y}(l)$ and $\mathbf{R}(j|t)$ are all independent of j as long as $j \le N - P$, that is, as long as there are more tasks available than there are processors. (In what follows, we drop the dependence on j when $j \le N - P$. E.g., $\mathbf{B}(l) = \mathbf{B}$ in that range.) Even so, the $\mathbf{p}(j)$'s, $\mathbf{R}(j|t)$'s and $T(j)$'s only approach a constant value with increasing j, assuming N is large enough. After all, from (7) we see that $\mathbf{p}(j) \ne \mathbf{p}(j+1)$, and they only approach each other as they approach the steady-state vector, $\mathbf{p}$, defined by

$$\mathbf{p} = \lim_{j,N\to\infty} \mathbf{p}(j) = \mathbf{pY} \qquad (9)$$

where $\mathbf{Y} = \mathbf{Y}(l)$ for every $l \ge P$. Note that once $l < P$ all the matrices change with l. In the last P epochs everything changes. Even so, we will call the region $j_0 < j \le N - P$ the *steady-state (s.s.) zone* (if it exists), where j_0 is big enough so that the variation in the $\mathbf{p}(j)$'s is negligible. The region below j_0 is called the

transient zone, and we will call the region, $1 < P$ ($N - P < j \le N$) the *completion zone* of the job. For the mean completion time for the j^{th} epoch, from the above discussion, it should be clear that when $N = P$, there is *no* s.s. zone. Fig. 2 illustrates the above behaviors when $N >> P$ and $N = P$.

From the discussion preceding (4), it follows that $[\mathbf{pV}]_k$ is the mean time that there are k functional processors during any epoch in the steady-state zone, and since this doesn't change from epoch to epoch, the s.s. probability of finding k functional processors (in the s.s. zone) is

$$\Pr(K = k) = \frac{[\mathbf{pV}_k]}{T},$$

where $T = \mathbf{pV}\varepsilon'$ is the s.s. value of $T(j)$ from (4).

We see then, that the status of the hardware (available processors) is decoupled from the task-stream in the s.s. zone, but not in the transient zone. Some researchers integrate the Chapman-Kolmogorov equations from $x = 0$ to the s.s. zone to find the availability of the system there. The *total availability*, which in our model would be, from (8),

$$\mathrm{A}(N) = \sum_{j=1}^{N} A(j) = (N - P)\tau + \sum_{j=N-P}^{N} A(j).$$

For finite N, the completion zone is treated incorrectly if the hardware is decoupled from the task-stream. After all, now there are fewer tasks than there are processors, whether they are available or not. This is particularly significant if only active processors can fail. If N is not sufficiently large so that the s.s. zone is insignificant, or even non-existent, then the availability of processors will not tell a proper story.

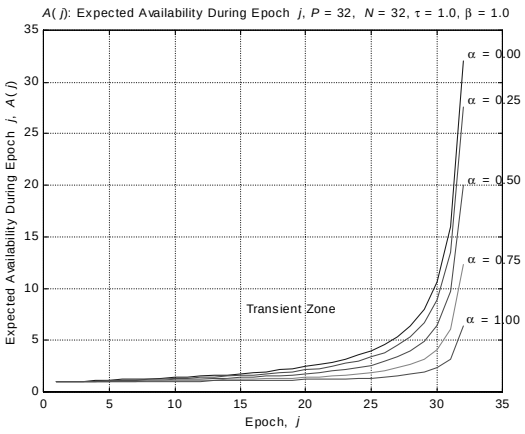


Fig 3. - The expected time processors are available during the j^{th} epoch, $A(j)$, when $P = 32$, $N = 22$, $\beta = 1.0$, and $\tau = 1.0$ for a between 0.00 to 1.00 inclusive.

Recalling when $N = P$, there exists no s.s. zone, then from the viewpoint of the task-stream, the system is *always* in the transient zone. Consequently, decoupling the availability from the workload in this case will likely lead to misleading measures regarding system availability. This behavior is demonstrated in Fig. 3.

As a final comment, we would expect the coupling of hardware to task-stream to be even more important for systems where there are arrivals as well as departures, for then the system will be continually moving among the three zones.

5. AN EXAMPLE

Let us consider the simplest non-trivial example for any N. First we mention that $P = 1$ (a 'trivial' case) is a straight-forward renewal process, where each epoch is generated by the < **p**, **B**> pair:

$$\mathbf{p}(j) = \mathbf{p} = [1, 0]; \quad \mathbf{B}(l) = \mathbf{B} = \begin{bmatrix} \alpha + \lambda & -\alpha \\ -\beta & \beta \end{bmatrix} \forall\ j.$$

That is all epochs, including the first and last are iid. The other quantities of interest are:

$$\mathbf{V}(l) = \mathbf{V} = \frac{\tau}{\beta}\begin{bmatrix} \beta & \alpha \\ \beta & \alpha + \beta \end{bmatrix}$$

and

$$\mathbf{Y}(l) = \mathbf{Y} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \forall\, l\,.$$

From (4), $T = \tau(1 + \gamma)$, where $\gamma = \alpha/\beta$ is the ratio of failure to repair rates. The availability vector (s.s. or otherwise) is

$$\mathbf{p_a} = \left[\frac{\beta}{\alpha + \beta}, \frac{\alpha}{\alpha + \beta}\right] = \left[\frac{1}{\gamma + 1}, \frac{\gamma}{\gamma + 1}\right].$$

The simplest non-trivial example is for $P = 2$. In this case

$$\mathbf{p}(1) = \mathbf{p} = [1, 0, 0]$$

and for $1 \le j < N$ (remember $l = N - j + 1$),

$$\mathbf{B}(l) = \mathbf{B}_2 = \begin{bmatrix} 2\lambda + 2\alpha & -2\alpha & 0 \\ -\beta & \alpha + \beta + \lambda & -\alpha \\ 0 & -\beta & \beta \end{bmatrix},$$

and

$$\mathbf{\Lambda}(l) = \mathbf{\Lambda}_2 = \lambda \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

However the last epoch is different, since there is only one task left, but there are 2 processors. So for $j = N$ ($l = 1$),

$$\mathbf{B}_1 = \mathbf{B}(l) = \begin{bmatrix} \lambda + z\alpha & -z\alpha & 0 \\ -\beta & \alpha + \beta + \lambda & -\alpha \\ 0 & -\beta & \beta \end{bmatrix},$$

and

$$\mathbf{\Lambda}_1 = \mathbf{\Lambda}(l) = \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

We are considering two cases together. For $z = 1$ only active processors can fail, while $z = 2$, idle processors can fail as well. Note that in all cases, $\mathbf{B}(l)\varepsilon' = \mathbf{\Lambda}(l)\ \varepsilon'\ \forall\ l$.

From its definition, it can be shown that

$$\mathbf{V}_2 = [\mathbf{B}_2]^{-1} = \frac{1}{B_2} \begin{bmatrix} \beta(\beta+\lambda) & 2\alpha\beta & 2\alpha^2 \\ \beta^2 & 2\beta(\alpha+\lambda) & 2\alpha(\alpha+\lambda) \\ \beta^2 & 2\beta(\alpha+\lambda) & 2[(\alpha+\beta)^2 + \beta\lambda] \end{bmatrix}, \quad (10)$$

where $B_2 = |\mathbf{B}_2| = 2\beta\lambda(\alpha + \beta + 1)$ is the determinant of $\mathbf{B}_2$. Furthermore,

$$\mathbf{Y}_2 = \mathbf{V}_2\mathbf{\Lambda}_2 = \frac{1}{\alpha+\beta+\lambda} \begin{bmatrix} \beta+\lambda & \alpha & 0 \\ \beta & \alpha+\lambda & 0 \\ \beta & \alpha+\lambda & 0 \end{bmatrix}. \quad (11)$$

Additional calculations yield (for $\mathbf{pY}_2 = \mathbf{p}$)

$$\mathbf{p} = \left[\frac{\beta}{\alpha+\beta}, \frac{\alpha}{\alpha+\beta}, 0\right]; \qquad T_2 = p\mathbf{V}_2\varepsilon' = \tau\frac{1+2\gamma+2\gamma^2}{2(1+\gamma)} \quad (12)$$

and

$$\mathbf{p}_a = \frac{pV2}{pV2\varepsilon'} = \frac{[\beta^2, 2\alpha\beta, 2\alpha^2]}{\beta^2, 2\alpha\beta, 2\alpha^2}. = \frac{[1, 2\gamma, 2\gamma^2]}{1, 2\gamma, 2\gamma^2}.$$

$(\mathbf{p}_a)_j$ is the s.s. probability that a random observer will find j processors available, and is the s.s. solution of the M/M/2/2 queue. T_2 is the mean time per epoch when the system is in the s.s.

The other matrices of interest are:

$$\mathbf{Y}_1 = \mathbf{V}_1\mathbf{\Lambda}_1 = \frac{1}{z\alpha+\beta+\lambda} \begin{bmatrix} \beta+\lambda & z\alpha & 0 \\ \beta & z\alpha+\lambda & 0 \\ \beta & z\alpha+\lambda & 0 \end{bmatrix}.$$

What we want from these quantities are T(N) and A(N), and then examine them to see how they depend upon the task-stream, as represented by τ (or $1/\lambda$) and N. From (4), (5), and (7) we have

$$\begin{aligned} T(N) &= \sum_{j=1}^{N-1} T(j) + T(N) = \sum_{j=1}^{N-1} \mathbf{p}(j)\mathbf{V}_2\varepsilon' + \mathbf{p}(N)\mathbf{V}_1\varepsilon' \\ &= \mathbf{p}(1)\left[\sum_{j=1}^{N-1} \mathbf{p}(j)\mathbf{Y}_2\right]\mathbf{V}_2\varepsilon' + \mathbf{p}(1)\mathbf{Y}_2^{N-1}\mathbf{V}_1\varepsilon'. \end{aligned} \quad (14)$$

We can give a closed-form expression for this by finding the spectral decomposition of $\mathbf{Y}_2$ which we now do. First, the *Spectral Decomposition Theorem* states that any matrix $\mathbf{S}$, can be written in the following form:

$$\mathbf{S} = \sum \lambda_n \mathbf{v}_n \mathbf{u}_n,$$

where

$$\mathbf{u}_n\mathbf{S} = \lambda_n\mathbf{u}_n \quad \text{and} \quad \mathbf{S}\mathbf{v}'_n = \lambda_n\mathbf{v}'_n, \quad (15)$$

is normalized to satisfy $\mathbf{u}_n \cdot \mathbf{v}_n' = 1$. (Recall that objects like $\mathbf{v}_n' \cdot \mathbf{u}_n$ are square matrices of rank 1.) That is, { λ_n }, { $\mathbf{u}_n$ }, and { $\mathbf{v}_n'$ } are the set of eigenvalues, left eigenvectors, and right eigenvectors respectively, of $\mathbf{S}$. It is not hard to show that $\mathbf{u}_k \cdot \mathbf{v}_k' = 0$ for $n \neq k$. Therefore,

$$\mathbf{S}^j = \sum_{n=1}^{m} \lambda_{n_n}^j \mathbf{v}'_n \mathbf{u}_n, \quad \text{for} \quad j \geq 0. \quad (16)$$

Since $\mathbf{Y}(l)\varepsilon' = \varepsilon'$, 1 is an eigenvalue of $\mathbf{Y}(l)$ with right eigenvector ε'. All $\mathbf{Y}(l)$'s also share the property that their last column are all 0's ($\mathbf{Y}(l)_{0i} = 0$, $\forall\ i$). Consequently, they all have at least one eigenvalue of 0.

We now turn our attention specifically to $\mathbf{Y}_2$ from (11) to find an explicit expression for T(N) in (14). Solving for $\|\mathbf{Y}_2 - \lambda\mathbf{I}\| = 0$ yields three eigenvalues, $\lambda_0 = 0$, $\lambda_1 = 1$, and

$$\lambda_{2\cdot} = \frac{\lambda}{\alpha+\beta+\lambda}.$$

Solving for the eigenvectors from (15) and substituting them into (16) where $\mathbf{S} = \mathbf{Y}_2$ we get

$$\mathbf{Y}_2{}^j = \begin{bmatrix}1\\1\\1\end{bmatrix}\left[\frac{1}{1+\gamma}, \frac{\gamma}{1+\gamma}, 0\right] + \delta_{0j}\begin{bmatrix}1\\1\\1\end{bmatrix}[0,-1,1] + (\lambda_2)^j\begin{bmatrix}-\frac{\gamma}{1+\gamma}\\ \frac{\gamma}{1+\gamma}\\ \frac{\gamma}{1+\gamma}\end{bmatrix}[-1,1,0],$$

where δ_{ij} is the Kronecker delta function, which equals 1 for $i = j$, and is 0 otherwise. We next need to evaluate

$$\mathbf{X}(N) = \sum_{j=1}^{N-1} \mathbf{Y}_2{}^{j-1}.$$

This is easy enough since $\sum_{j=1}^{N-1}(1) = N-1$, $\sum_{j=1}^{N-1}\delta_{0,j-1} = 1$ and

$$\sum_{j=1}^{N-1}(\lambda_2)^{j-1} = \frac{1-\lambda_2^{N-1}}{1-\lambda_2} = \frac{\alpha+\beta+\lambda}{\alpha+\beta}\left[1-\left(\frac{\lambda}{\alpha+\beta+\lambda}\right)^{N-1}\right].$$

Putting this altogether, we get

$$\mathbf{X}(N) = (N-1)\varepsilon'\mathbf{p} + \begin{bmatrix}0\\0\\1\end{bmatrix}[0,-1,1] + \frac{\alpha+\beta+\lambda}{\alpha+\beta}\left[1-(\lambda_2)^{N-1}\right]\begin{bmatrix}-\frac{\gamma}{1+\gamma}\\ \frac{\gamma}{1+\gamma}\\ \frac{\gamma}{1+\gamma}\end{bmatrix}[-1,1,0].$$

The three terms each have their own meaning. The first term provides the time as though the system is always in the s.s. The second term only contributes if the system happens to begin with all processors down, and the third term provides the (initial) transient contribution.

We relieve the reader of the burden of going through the rest of the calculation. Suffice to say that if one (carefully) places this expression, together with (10) and (13) into (14), one gets

$$\mathrm{T}(N) = (N-1)T_2 + T_1 - \frac{\alpha(2\alpha+\beta)}{2\beta(\alpha+\beta)^2} + \frac{(z\alpha+\lambda)(2\alpha+\beta)+\beta^2-2\alpha^2}{2\beta(\alpha+\beta)^2(z\alpha+\beta+\lambda)}\lambda_2^{N-1} \tag{17}$$

where T_2 is given in (12) and

$$T_1 = \mathbf{p}\mathbf{V}_1\varepsilon' + \frac{\beta^2(\alpha+\beta)+(\alpha+\beta)^2+\lambda(\beta^2+\alpha\beta+\alpha^2)}{\beta\lambda(\alpha+\beta)(z\alpha+\beta+\lambda)}$$

is the mean time for the last task (completion zone) if the s.s. period had previously been reached (i.e., N is large enough so that $\mathbf{p}(1)\mathbf{Y}_2{}^{N-1} = \mathbf{p}$). Any deviation from this is included in the term containing $(\lambda_2)^{N-1}$.

These are certainly too cumbersome expressions from which to gain direct insight. But first notice that if $\alpha = 0$ (no failures), $T_2 = \tau/2$; $T_1 = \tau$, and T(N) = $\tau(N+1)/2$ as would be expected. The same results occur if $\beta \to \infty$ (instant repair).

We net examine the limit of T(N) if $W = \tau N = N/\lambda$ is held constant as $N \to \infty$ and $t \to 0$ (or $\lambda \to \infty$). First we look at $(\lambda_2)^{N-1}$

$$(\lambda_2)^N = \left[\frac{\lambda}{\alpha+\beta+\lambda}\right]^N = \left[1+\frac{\alpha+\beta}{\lambda}\right]^{-W\lambda}.$$

It follows then from elementary calculus that

$$\lim_{\lambda\to\infty}(\lambda_2)^N = e^{-(\alpha+\beta)W}.$$

The rest follows easily, yielding:

$$\lim_{\lambda\to\infty}\mathrm{T}(N) = W\frac{1+2\gamma+2\lambda^2}{2(1+\gamma)} - \frac{\gamma(1+2\gamma)}{2\beta(1+\gamma)^2}\left(1+e^{-(\alpha+\beta)W}\right).$$

The first term on the right is the time the job would take if there were no transient effects, while the second term gives the transient contribution, which reduces to total execution time because initially all processors are functional. The completion zone doesn't contribute in this limit, because only one job

remains at the end, and $t \rightarrow 0$. The formula also tells us that the s.s. region occurs if $(\alpha + \beta)W >> 1$ (the exponential term goes to 0), but even then, the effects of the transient region can be significant unless W is very large.

6. CONCLUSIONS

There has been much research in the area of the design, implementation, and performance analysis of computing systems in the presence of failures and repairs. In this paper, we discussed an analytic procedure for evaluating the availability of a computer system comprised of P processors subject to failures and repairs. In addition, by using our approach, we claimed the state space is reduced by a factor of N compared to other techniques (see [13]). In addition, via our analytic and numeric examples, we argued that not incorporating the task-stream is an inadequate approach for evaluating system performance. Furthermore, we stated this would especially be true in interactive environments where both arrivals and departures could occur since the system would be continuously moving between the transient, steady-state, and completion zones. Furthermore, we showed that when $N = P$, from the viewpoint of the task-stream, the system is always in the transient zone, thus, decoupling availability from the workload in this case would likely lead to misleading performance results.

7. REFERENCES

[1] L. Donatiello and B. R. Iyer. Analysis of a Composite Performance Reliability Measure for Fault-Tolerant Systems, Journal of ACM 34 (1) (1987). p. 179-199.

[2] B. Havakort, R. Marie, G. Rubino, and K. Trivedi. Performability Modeling: Tools and Techniques. Editors. Wiley, New York, 2002.

[3] P. Kanellakis and A. Shvartsman. Fault-Tolerant Parallel Computation. Kluwer Academic Publishers, Boston, 1997.

[4] J.Meyer. On Evalutating the Performability of Degradable Computing Systems, IEEE Transactions on Computers (C-29) 8 (1980).

[5] I. Mitrani and A. Puhalskii. Limiting Results for Multiprocessor Systems with Breakdowns and Repairs, Queueing Systems (14) (1993).

[6] I. Mitrani and P.E. Wright. Routing in the Presence of Breakdowns, Performance Evaluation (20) (1994).

[7] H. Nabli and B. Sericola Performability Analysis: A New Algorithm, IEEE Transactions on Computers, 45 (4) (1996).

[8] M.F. Neuts. Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach, Johns Hopkins University Press, Baltimore, 1981.

[9] S. Osaki and T. Nishio. Reliability of Some Fault-Tolerant Computer Architectures, Springer-Verlag, New York, 1980.

[10] R.A. Sahner, K.S. Trivedi, and A. Puliafito. Performance and Reliability Analysis of Computer Systems: An Example-Based Approach using the SHARPE Software Package, Kluwer Academic Publishers, Boston, 1996.

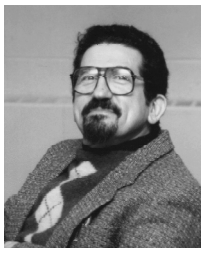
[11] K. Wolter and A. Zisowsky. On Markov Reward Modeling with FSPN's. Proceedings of "4th International Computer Performance and Dependability Symposium", Chicago, IL, March 2000.

[12] K. Trivedi, J. Muppala, S. Woolet, and B. Havakort. Composite Performance and Dependability Analysis, Performance Evaluation (1992).

[13] A. Bobbio and K. Trivedi. Computation of the Distribution of the Completion Time When the Work Requirement is a PH Random Variable, Communications in Statistics - Stochastic Models, (6) 1 (1990).

[14] M.F. Neuts. Structured Stochastic Models of the M/G/1 Type and their Applications, Marcel Dekker, New York, 1989.

Pierre M. Fiorini *is an Assistant Professor of Computer Science at the University of Southern Maine. He received the Ph.D. degree from the University of Connecticut in Computer Science & Engineering (1998), an M.S. in Computer Science & Engineering from the University of Connecticut (1995), and a B.S. in Computer Science from Trinity College (1989). His research interests include Queueing Theory, Computer Performance Modeling, Network Modeling, Stochastic Processes, and Computational Intelligence. He is a member of the IEEE and ACM.*



Lester Lipsky *is a Professor of Computer Science & Engineering at the University of Connecticut. He holds the Ph.D. degree from the University of Connecticut in Theoretical Atomic Physics (1965), an M.S. in Physics from Brandeis University (1958), and a B.M.E in Mechanical Engineering from the City College of New York (1956). His research Interests include: Queuing Theory (Linear Algebraic Approach), Computer System Performance Modeling, Network Modeling, Stochastic Processes Related to Telecommunications, and Computational Atomic Physics. He is the author the text entitled, Queueing Theory: A Linear Algebric Approach (MacMillan & Co.) and a member of the IEEE, ACM, Sigma Chi, Upsilon Pi Epsilon, and the American Physical Society.*