

# Approximate analysis of a fault-tolerant join-the-shortest-queue policy

Guangzhi Li    Gianfranco Ciardo  
Department of Computer Science  
College of William and Mary  
Williamsburg, VA 23187-8795, USA  
{gli, ciardo}@cs.wm.edu

## Abstract

*In distributed or multi-processor systems, the join the shortest queue (JSQ) scheduling policy can provide competitive performance compared to other heuristic policies. However, processing nodes can fail. In order to provide fault tolerance, one proposed solution is using a “buddy system”. We investigate a new scheduling system by combining these two methods together to provide good performance and high reliability. In this paper, we describe the system in detail and evaluate its performance using an approximation based on the matrix-geometric method. Our results show that our method provides very accurate estimates for the average number of jobs in the system.*

## 1. Introduction

We consider the following system: in a computer center, there are  $N$ ,  $N \geq 2$ , identical servers. Each server has a queue with infinite capacity. There is a job dispatcher responsible for assigning the incoming jobs to servers. The job arrival process to the center is assumed to be Poisson with rate  $\lambda$ . The job service time is assumed to be exponentially distributed with mean  $1/\mu$ . Servers can fail and be repaired, independently of each other. The server failure and repair times are also exponentially distributed, with mean  $1/\delta$  and  $1/\gamma$ , respectively.

When a job arrives, we use the “join the shortest queue” (JSQ) policy, that is, the job is assigned to the server with the smallest number of jobs in its queue. If multiple servers have the minimum number of jobs, ties are broken by uniformly choosing one of them. In our analysis, we assume that there is no jockeying of customers among queues. This policy has been often used as a basic load-balance or routing mechanism in distributed or multi-processor systems [6, 7, 8].

An approximate analysis of the JSQ policy can be found in [1], which does not consider server failures. A birth-

death Markov process is used to model the evolution of the number of jobs in the system. The transition rates are estimated using an iterative procedure. To show the accuracy of this approximation method, estimates of the average job response times are compared with simulation results, with excellent agreement. More results about the analysis of the JSQ policy can be found in [7, 8].

However, a server may fail and, when it does, the jobs in its queue are lost. A simple scheme called “buddy system” has been defined to provide fault-tolerance [2] in a multi-processing environment. In a buddy system, each working server has exactly one buddy, and it is buddy to exactly one other working server. When a server receives a “regular” job, it sends a “copy job” to its buddy. The buddy holds this copy in a copy queue, separate from its own regular queue. If the original server fails, the buddy moves all the jobs from its copy queue to its regular queue, and send copies of them to its own buddy. This provides  $(N - 1)$ -failure fault-tolerance, if we ignore the possibility of a failure while jobs are in transit to or from a server (i.e., if we assume no near-coincident faults). Of course, every time a server completes a regular job, it sends a message to its buddy, so that the corresponding copy job can be deleted. In a practical implementation, the buddy must periodically send a message to its sending server to determine whether it has failed. Analogously, if a server stops receiving these messages from its buddy, it can assume its buddy has failed; then, it must find a new buddy and send copies of all its regular jobs to this new buddy.

Algorithms to redistribute the workload of failed nodes to the remaining working processors have been analyzed in [9]. The authors construct a Markov chain model of the load distribution problem and obtain bounds on the performance of several load distribution algorithms. In [2], an approximate model for computing the queue length of the regular jobs in the buddy system is presented. A testbed of the buddy system has been implemented on a network of four AT&T 3B2 minicomputers [4].

In our study, we combine the JSQ policy and the buddy

scheme, to provide good performance and high reliability. We assume that the  $N$  servers can communicate directly with each other, so that the particular choice of the buddy is statistically not important. Thus, when a server fails, its buddy dispatches copies of the failed server's jobs, to any server (including itself) according to the JSQ policy. Of course, a server receiving such a job treats it as an ordinary regular job assigned to it, that is, it sends a copy to its own buddy.

To analyze the proposed system, we develop an exact Petri net simulation model of the entire system and obtain the average number of jobs in the system. However, simulation is expensive. On the other hand, a traditional exact numerical solution of the underlying Markov chain is impossible, since the state space for the Petri net is infinite. A truncation of the state space is possible in principle, by assuming that the population in each queue never exceeds  $T$ . However, especially for large values of  $N$  and fairly loaded systems, the value of  $T$  for which good approximations can be obtained is too large, since the size of the state space grows as  $T^N$ .

Hence, we propose an approximation that uses a G/M/1-type Markov process to model the evolution of the total number of jobs in the system and the total number of working servers. The matrix-geometric approach is used to solve our G/M/1-type Markov process, and a fix-point iteration scheme is employed to estimate the service rates in our macro-states. Upon convergence, we can obtain the average number of jobs in the system. Comparison with the results from our Petri net simulations show that our method provides very accurate estimates.

## 2. Petri net model

The stochastic Petri net (SPN) in Fig. 1 models our system. Jobs arrive by firing transition  $A$  and enter place  $jsq$ . Then, jobs are distributed to one of the working servers by firing one of the immediate transitions  $t_1, t_2, \dots, t_N$ , say  $t_i$ , and enter into the corresponding waiting place  $W_i$ . There are  $N$  servers, which can fail and be repaired, by firing transition  $F_i$  and  $R_i$ , respectively. If transition  $F_i$  fires, all jobs in place  $W_i$  are moved to place  $jsq$  and redistributed onto the other working servers (the arcs from  $W_i$  to  $F_i$  and from  $F_i$  to  $jsq$  are bold to denote this “flushing” effect). If all servers are down, the “guard”  $g$  of transition  $L$  is true, and  $L$  can fire and remove all jobs in place  $jsq$ . Each transition  $t_i$  has a guard as well: for  $t_i$  to be enabled, the number of jobs in place  $W_i$  must be (one of) the smallest among all waiting places with a working server. The firing rates of  $A$ ,  $S_i$ ,  $F_i$ , and  $R_i$  are  $\lambda$ ,  $\mu$ ,  $\delta$ , and  $\gamma$ , respectively.

In Fig. 1, we assume that servers are connected by a very fast network, so that job migration and message passing require essentially no time, and we can model these actions

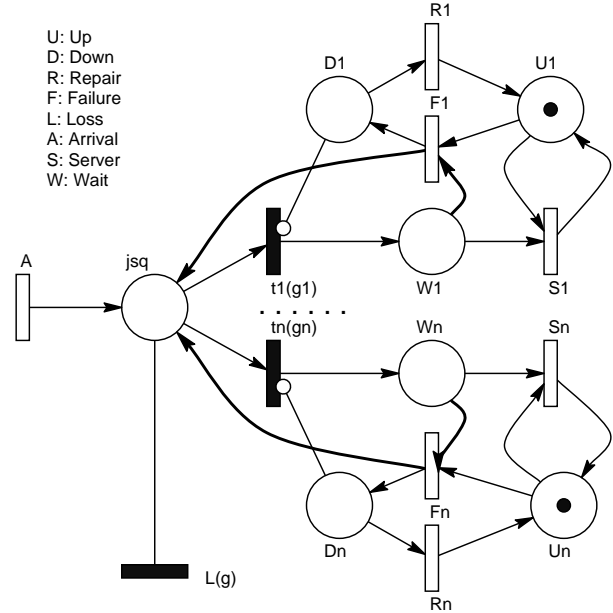


Figure 1. Petri net model.

with the immediate transitions  $t_1, t_2, \dots, t_n$ . In reality, job migration (especially for a running job migration) is a very complex problem, and the possibility of a failure occurring during migration would require additional safeguards.

## 3. Approximate model

We can represent the complete state of the system with a vector of size  $N$ , where the  $k^{\text{th}}$  entry is set to the number of jobs in queue  $k$ , if server  $k$  is working, or to  $-1$ , if it is failed. For a given total number of jobs  $j$  in the system and a number of working servers  $i$ ,  $1 \leq i \leq N$ , the  $j$  jobs may be distributed over the  $i$  working server queues in any way, each of which represents a different state of the system.

To reduce the size of the state space, we combine states with the same number of working servers and the same total number of jobs in the system into a single state in our approximate model. Then, the system evolution can be approximated by a Markov process of the G/M/1-type [3], whose two-dimensional state space is shown in Fig. 2. In state  $(i, j)$ , the job arrival rate is  $\lambda$ , the total server failure rate is  $i\delta$ , the total server repair rate is  $(N - i)\gamma$ , and the average service rate is  $\mu(i, j)$ . The first three parameters are easy to compute, but we need to estimate the fourth parameter  $\mu(i, j)$ , representing the average service rate of the system in state  $(i, j)$ , that is, when there are  $i$  working servers and a total of  $j$  jobs.

The procedure for estimating the values  $\mu(i, j)$  is iterative. At each iteration, the service rate for state  $(i, j)$  is

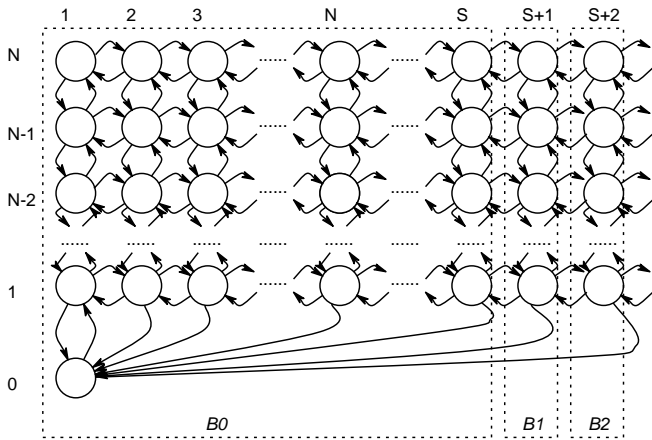


Figure 2. The G/M/1-type Markov process.

obtained by conditioning on the state of the system before entering  $(i, j)$ . Hence, we use the probability of its neighboring states at the previous iteration, as weights. If state  $(i, j)$  was entered from state  $(i, j + 1)$ , the most recent state transition must have been a job departure, which could have left an empty or nonempty queue behind; in the former case the service rate was reduced by  $\mu$ , in the latter it remained unchanged. If  $(i, j)$  was entered from state  $(i, j - 1)$ , the most recent state transition must have been a job arrival, and there may or may not have been an idle working server; in the former case the service rate was increased by  $\mu$ , in the latter it remained unchanged. If  $(i, j)$  was entered from state  $(i - 1, j)$ , the most recent transition must have been the repair of a failed server, and the average service rate remained unchanged. If  $(i, j)$  was entered from state  $(i + 1, j)$ , the most recent transition must have been caused by a server failure, the failed server may or may not have had jobs, and the other working servers may or may not have had idle servers; if the failed server or one of the other servers was idle, the service rate remains unchanged, otherwise it is decreased by  $\mu$ .

First,  $\mu(i, 1) = \mu(1, j) = \mu, \forall i > 0, j > 0$ , because there is one and only one working server busy. If there are more than one jobs and more than one working servers, we need to know the distribution of the number of jobs to the working servers. Note that at least one of the servers must be busy if  $i > 0$  and  $j > 0$ . As in [1], we approximate the distribution of the number of busy servers minus one using a binomial distribution with mean equal to the average number of busy servers minus one. Specifically, in state  $(i, j)$ , the binomial has two parameters  $B(n, p)$ . We let  $n = i - 1$ , and the mean  $np = \mu(i, j) - 1$ . When the total number of jobs is larger than the number of busy servers, we assume the waiting jobs to be distributed evenly over the largest possible number of nonempty queues since the na-

ture of the JSQ is to try to balance the queue length.

When the number of jobs in the system becomes relatively large, say  $S$ , we expect the number of busy servers to be very close to the number of working servers since the number of jobs in each of their queues is most likely nonzero. Therefore, for state  $(i, j)$  with  $j \geq S$ , we approximate the service rate with  $i\mu$ . The value of  $S$  is chosen such that the difference between  $\mu(N, S)$  and  $N\mu$  is small enough, (less than  $10^{-6}$ ). In fact,  $\mu(i, j), i > 1$  will always be less than  $i\mu$ , no matter what  $j$  is, because there is always a nonzero probability that a server was just repaired and its queue is empty. Since the server failure rate is much smaller than the job arrival rate, the difference between  $\mu(i, j)$  and  $i\mu$  will be very small for large values of  $j$ . However, this shows that our estimations are likely to be optimistic.

Given the service rates at the  $k^{\text{th}}$  iteration and the corresponding stationary probabilities, we can compute the service rates at the  $(k + 1)^{\text{th}}$  iteration using the above approximation. We then iterate this process in a fixed-point fashion, until our estimates for the service rates remain stable within a certain tolerance. All that remains to be discussed are the estimation of  $S$  and the computation of the stationary probabilities for our G/M/1-type Markovian process, for a given choice of service rates. We do these in the next two sections.

## 4 Estimation of $S$

Recall that, when the number of jobs in the system becomes relatively large, say  $S$ , we expect the number of busy servers to be very close to the number of working servers since the number of jobs in each of their queues is most likely nonzero. The question is how to estimate  $S$ . Note that, if the probability that all servers are busy is very close to one when there are  $S$  jobs and  $N$  servers, this is even more so when there are  $S$  jobs and fewer than  $N$  servers. Hence, in a sense,  $N$  is a safe worst-case assumption when deciding a value for  $S$ . So, we choose  $N$  servers without failure and construct a birth-death Markovian chain: state  $i$  stands for  $i$  jobs in the system and  $N$  servers.

First, we guess  $S = 2N$ . Based on the same binomial assumption of  $i$  jobs distributed among  $N$  servers and using an iterative approach, the average service rate  $\mu(i)$  is updated using the probabilities of the possible previous states. If the previous state was  $i - 1$ , the last transition must have been an arrival, and there may or may not have been an idle server; in the former case, the service rate was increased by  $\mu$ , in the latter it remained unchanged. If the previous state was  $i + 1$ , the last transition must have been a job departure, which may or may not have left an empty server; the former case leads to an average service rate decreased by  $\mu$ , while the latter leaves it unchanged. From these conditional probabilities and the closed-form steady-state probability distri-

bution of the birth-death Markov chain, we can compute the average service rate  $\mu(i)$  in this system with no failures. If  $|\mu(S) - N\mu|$  is small enough (less than  $10^{-6}$ ), we have an estimation of  $S$ , otherwise we double  $S$  and repeat the above procedure. In our experiments, this estimation process requires only few iterations.

## 5. Stationary analysis

We assume that readers are familiar with the matrix-geometric approach. For more information, see [3].

To order the states of our Markov process sequentially, we assign index  $i + j \cdot N$  to state  $(i, j)$ . Then the state space  $\mathcal{S}$  can be partitioned into two categories: the boundary states  $\mathcal{B}_0 = \{0, 1, \dots, S \cdot N\}$  and the set of states  $\mathcal{B}_1 = \{S \cdot N + 1, \dots, (S + 1) \cdot N\}, \mathcal{B}_2 = \{(S + 1) \cdot N + 1, \dots, (S + 2) \cdot N\}, \dots$  corresponding to the repetitive portion of the chain. The infinitesimal generator  $Q$  of our G/M/1-type Markov process can be expressed as:

$$Q = \begin{bmatrix} L^{(00)} & F^{(01)} & 0 & 0 & 0 & \dots \\ B^{(10)} & L & F & 0 & 0 & \dots \\ A & B & L & F & 0 & \dots \\ A & 0 & B & L & F & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

where  $L^{(00)} \in \mathbb{R}^{(NS+1) \times (NS+1)}$  represents the transition rates between states in  $\mathcal{B}_0$  (failures, repairs, arrivals, and departures),  $F^{(01)} \in \mathbb{R}^{(NS+1) \times N}$  represents the transition rates from states in  $\mathcal{B}_0$  to states in  $\mathcal{B}_1$  (arrivals when there are  $S$  jobs already),  $B^{(10)} \in \mathbb{R}^{N \times (NS+1)}$  represents the transition rates from states  $\mathcal{B}_1$  to states in  $\mathcal{B}_0$  (departures when there are  $S + 1$  jobs, or failure of the last working server),  $F \in \mathbb{R}^{N \times N}$  represents transition rates from states in  $\mathcal{B}_j$  to states in  $\mathcal{B}_{j+1}$  (arrivals),  $L \in \mathbb{R}^{N \times N}$  represents transition rates between states in  $\mathcal{B}_j$  (failures that leave some working server and repairs),  $B \in \mathbb{R}^{N \times N}$  represents transition rates from states in  $\mathcal{B}_j$  to states in  $\mathcal{B}_{j-1}$  for  $(j > 0)$  (departures), and  $A \in \mathbb{R}^{N \times (NS+1)}$  represents the transition rates from states in  $\mathcal{B}_j$  to states in  $\mathcal{B}_0$  (failures of the last working server).

Let  $\pi^{(j)}$  be the stationary probability vector for states in  $\mathcal{B}_j$ , for  $j \geq 0$ . Then, the stationary probability vector  $\pi$  can be partitioned as  $\pi = [\pi^{(0)}, \pi^{(1)}, \dots]$ .

According to [3], we can hypothesize the existence of a relation of the form  $\pi^{(j)} = \pi^{(j-1)} \cdot G$ , for  $j \geq 2$  where  $G$  is the solution of the quadratic matrix equation

$$F + G \cdot L + G^2 \cdot B = 0.$$

Using the equality  $\pi^{(j)} = \pi^{(1)} \cdot G^{j-1}$ , we can then obtain a linear system in  $\pi^{(0)}$  and  $\pi^{(1)}$ :

$$[\pi^{(0)}, \pi^{(1)}] \cdot Q' = 0,$$

where

$$Q' = \begin{bmatrix} L^{(00)} & F^{(01)} \\ B^{(10)} + G \cdot (I - G)^{-1} \cdot A & L + G \cdot B \end{bmatrix}.$$

It is easy to prove that  $Q'$  is an infinitesimal generator matrix. So, considering the normalization condition  $\sum_{i=0}^{\infty} \pi^{(j)} \cdot \mathbf{1}^T = 1$ , we can compute a unique solution. Once  $\pi^{(0)}$  and  $\pi^{(1)}$  have been computed, any other  $\pi^{(j)}$ ,  $j \geq 2$ , can be obtained numerically. In fact, the average number  $L$  of jobs in the system can be calculated exactly without having to compute  $\pi^{(j)}$  explicitly for any other value of  $j$ , as follows:

$$L = \sum_{k=1}^{S-1} k \left( \sum_{i=1}^N \pi_{(kN+i)}^{(0)} \right) + S \cdot \pi^{(1)} \cdot (I - G)^{-1} \cdot \mathbf{1}^T + \pi^{(1)} \cdot G \cdot (I - G)^{-2} \cdot \mathbf{1}^T.$$

## 6. Algorithm

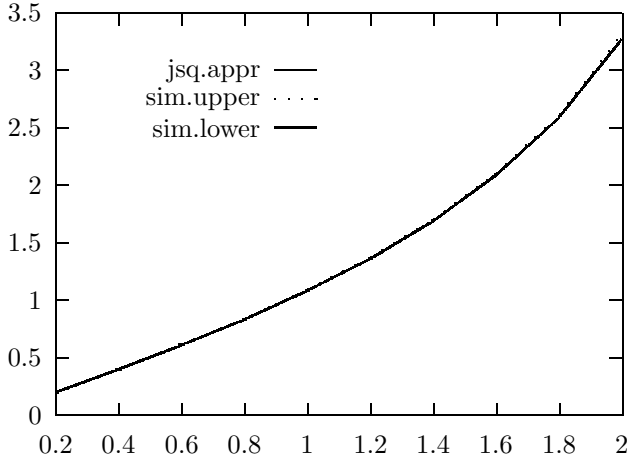
Our step-by-step computational procedure is:

1. Use a birth-death Markov process, assuming  $N$  working servers, to estimate the value of  $S$ .
2. Initialization:  $\mu(i, k) = \min(i, k)\mu$ ,  $\pi_{old}^{(0)} = \mathbf{0}$ .
3. Compute new values for the stationary probabilities,  $\pi_{new}^{(0)}$  and  $\pi_{new}^{(1)}$  using the matrix-geometric approach.
4. If  $\|\pi_{old}^{(0)} - \pi_{new}^{(0)}\|_{\infty} \leq 10^{-6}$ , go to step 6, else set  $\pi_{old}^{(0)} \leftarrow \pi_{new}^{(0)}$ .
5. Update the service rates with our heuristic. Go to step 3.
6. Compute the average number of jobs in the system.

## 7. Comparison with simulation

To study the accuracy of the analytical results, we use the SPNP simulator [5] to study our system with different parameters and measure the average number of jobs in the system. Figure 3 compares our approximation with the simulation results assuming the following parameters:  $N = 3$ ,  $\mu = 1.0$ ,  $\delta = 0.001$ ,  $\gamma = 0.1$ , and  $\lambda = 0.2, 0.4, \dots, 2.0$ . The x-axis represents the job arrival rate, while the y-axis stands for average number of jobs in the system. The plot labeled “jsq.appr” shows our the approximation result while “sim.upper” and “sim.lower” enclose the 95% confidence interval for the simulation.

Observe that the method we developed provides very accurate estimates of the average jobs in the system. In general, we tend to (very slightly) underestimate the average



**Figure 3. Our approximation vs. simulation.**

number of jobs in the system, because of two optimistic assumptions: waiting jobs are distributed evenly among the queues with all busy servers and, when there are more than  $S$  jobs in the system, all working servers are busy.

## 8. Conclusion

In this paper, we developed an approximation to analyze the performance of the Join the Shortest Queue (JSQ) policy when the “buddy system” is used to provide resilience against failures of the  $N$  servers. Both the job arrival and the server failure processes are assumed to be Poisson. Service time and server repair time are assumed to be exponentially distributed. We use a G/M/1-type Markovian process to model the evolution of the number of working servers and the number of jobs in the system. The transition rates are estimated in a fixed-point iterative fashion by assuming that the jobs are distributed over the working servers according to a binomial distribution. Each iteration requires a matrix-geometric solution of system with  $N \cdot (S + 1) + 1$  states, where  $S$  is large enough so that (almost) all working servers are busy when there are  $S$  jobs in the system. An exact Stochastic Petri net model of the system is simulated to study the accuracy of the method. The results show that our approach can estimate the average jobs in the system very accurately.

## References

- [1] H.C. Lin and C.S. Raghavendra, “An Approximate Analysis of the Join the Shortest Queue (JSQ) Policy”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 3, March 1996.
- [2] D. Finkel and S.K. Tripathi, “A Performance Analysis of a Buddy System for Fault Tolerance”, *Performance Evaluation* 11 (1990), 177-185.
- [3] M. Neuts, “Matrix-geometric solutions in stochastic models”, Johns Hopkins University Press, Baltimore, MD, 1981.
- [4] P. Dikshit, “SAHAYOG: A testbed for fault-tolerant loadsharing in a distributed system”, MS thesis, Department of Computer Science, University of Maryland, College Park, 1987.
- [5] G. Li, “A Stochastic Petri Nets Simulator Engine”, MS thesis, Department of Computer Science, College of William and Mary, 1998.
- [6] G. Foschini and J. Salz, “A Basic Dynamic Routing Problem and Diffusion”, *IEEE Transactions on Communications*, vol. 26, May 1978, 320-327.
- [7] C. Knessl, B. Matkowsky, Z. Schuss, and C. Tier, “Two Parallel M/G/1 Queues Where Arrivals Join the Systems with the Smaller Buffer Content”, *IEEE Transactions on Communications*, vol. 35, Nov. 1987, 1153-1158.
- [8] R.D. Nelson and T.K. Philips, “An Approximation to the Response Time for Shortest Queue Routing”, *ACM Performance Evaluation review*, vol. 17, May 1989, 181-189.
- [9] T.C. Chou and J.A. Abraham, “Load redistribution under failure in distributed systems”, *IEEE Transactions on Computers* 32(1983), 799-808.