Heterogeneous *k*-Server Fork/Join Blocking Queue Approximation with Deferred Job Pickup

Marc Mosko
Ovid Jacob

CMPE 230 Class Project
March 10, 1999

Marc Mosko
Ovid Jacob

CMPE 230 Class Project

Abstract We review the literature related to Fork-Join queues, as applicable to the class project. Based on several articles, we propose a simplified reduced state solution that only tracks the number of tasks in the fastest queue, the slowest queue, and the number of tasks waiting in the join area. This yields a continuous time Markov chain with three state variables. We model transitions caused by intermediate servers as a service rate times a probability that the server is busy. We base the conditional probability on the number of jobs in the fastest and slowest queues. Using an *m-Erlang* distribution for the time to complete *m* jobs, we estimate the probability that an intermediate queue is ahead of the fastest or slowest queues. We also present simulation results for the system over various system parameters.

Introduction

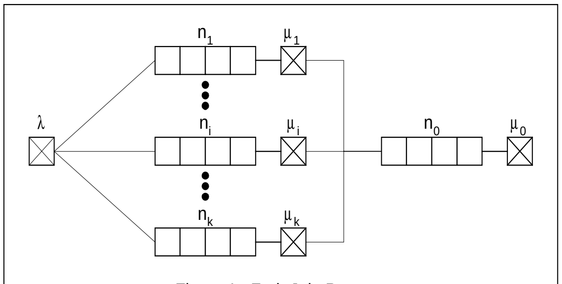


Figure 1. Fork-Join Process

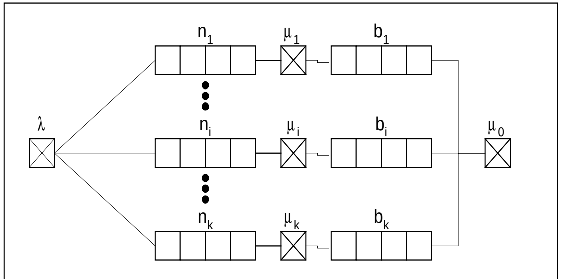


Figure 2. Fork-Join Process with Blocking [DA94]

We wish to model a Fork-Join process with k heterogeneous servers. We call the join point queue 0, with n_0 tasks and service rate μ_0. Unless otherwise noted, we picture the network as per Figure 1. In models proposed by Dallery [DA94], we use the structure of Figure 2. When a job arrives with Poisson rate λ, it is split into k identical tasks. After independent processing, tasks go to the join queue. When all k tasks of a job arrive, the job may be removed at a later time. Unlike normal fork-join models, completed jobs do not immediately depart the system. The model is restricted such that at any given time, there may be no more than N jobs in the system. With some periodicity Δ, a purge process runs and removes completed jobs from the join queue. We wish to compute the probability of removing *J* jobs given that there are *M* in the system.

We begin in Section 2 with a review of the literature as it applies to our problem. In Section 3 we present our model for the queuing system. We analyze the model in Section 4, commenting on robustness and stability. Section 5 presents our simulation methodology. Section 6 details the results of our model and simulations. We compare the results for certain system parameters. We summarize our findings in Section 7, concluding with thoughts on further research.

Section 2: Related Work

For two queues (*k=2*), one may find the exact solution to the non-blocking problem [FL84, FL85]. The solutions Flatto proposes are cumbersome and sophisticated. While one may use the results to validate an approximation [e.g. LU98, BA98], one may not easily adapt these equations to other purposes.

Dallery *et al.* [DA94] models Fork-Join networks with blocking using a graph method. His model uses blocking-before-service and requires an initial marking of the state of the system. As shown in Figure 2, Dallery's model uses an output buffer for each service facility. As applied to our problem, a given server would block service of jobs in the input buffer if there were not room in the output buffer. The purging process, shown as the sink server, would remove a number of jobs equal to the minimum of the jobs in the output buffer. The source server would block if there were no room in each of the input buffers. Using this model, we would need to adjust the input and output buffers as if they used shared memory. The total size of $n + b$ for a given server should exceed *N.* A simple approximation, which should hold for the equilibrium condition $\lambda < min(\mu_1 \dots \mu_k)$, would be to limit *n* to *N/2* and *b* to *N*. From our simulation results, blocking most often occurred from filling the join queue, not from overflowing the input queue. The removal process is more difficult to model.

One simple approximation of the removal process for Dallery's formulation is to give the sink a service rate of $min(b_1 \ldots b_k)/ \Delta$. This would give an "average" removal rate, but make the service rate at the sink state dependent. One might also try a service rate of N / Δ, and allow the model to starve the sink when there are fewer than *N* jobs waiting for pickup.

Dallery states a set of "evolution equations" which describe the time sequence of completions from each server. One could, in theory, use these equations to completely describe a system. As Dallery notes, it is difficult to use these equations for anything by deterministic service times. Otherwise, the service completion times become compound distributions that involve a "maximum" function. One finds compound distributions complex enough for simple Bernoulli and Poisson distributions.

In a subsequent paper [DA97], Dallery extends [DA94] to complex Fork/Join networks. These networks have various loading and unloading policies. [DA97] does not add any new constructs that would help us in our problem.

Varki *et al.* [VA96] use three equivalent models of a homogeneous fork/join network to derive performance characteristics. This model is not applicable to our situation of a blocked heterogeneous network. We will, however use a few of the ideas from Varki in our *m-Erlang* model. In a Markov model of a two-server system, Varki states that the arrival rate at the join point is equal to 2μ in the case both servers are busy or μ if only one server is busy. We shall use a similar idea, except with unequal service rates and an approximation of server busy periods.

Balsamo *et al.* [BA98] study approximations for a heterogeneous fork/join network similar to those found in parallel processing computer systems. Balsamo's state description is the number of jobs in each service queue. In her model, completed jobs at the join point immediately depart. There is no blocking. She constructs an infinite state Markov process, where an arrival bumps all queues to the *n+1* state. Departures affect individual queues and reduce them to the *n-1* state. This creates an embedded Markov chain in a continuous time Markov chain. Balsamo then uses two approximations to generate finite Quasi-

Birth/Death processes and proves that these provide an upper and lower bound the true solution. Balsamo shows that for a two server model, her approximation bounds the exact solutions found by Flatto [FL84, FL85]. Our solution is similar to Balsamo. Most notably, we reduce the state space further and do not use the Matrix-Geometric approach [NE81]. In our formulation of an *m-Erlang* system, we will make approximations to limit the state space to three degrees while accounting for blocking.

In another study that has some possible applicability to our problem, Frostig *et al.* [FR97] analyze stochastic ordering in fork-join queues. The main work studies a two queue system, but she later generalizes it to more servers. The infinite customer supply case only has a 1 dimensional state space: the number of departures. It leads to a set of three non-homogeneous Markov chain equations. Her state space for a starved system has four dimensions. She models the number of arrivals to the join queue, the number of departures from the join queue, the number of "lost" arrivals, and the choice of the fastest server. In a system with infinite customer supply, the system never loses an arrival. In a starved system, some arrivals from an infinite supply model are "lost" and cannot accumulate in the join queue. In the starved server case, the four dimensional state space has thirty-two equations covering different possibilities of state transitions. As one will see, we drew several ideas from this paper into our model.

Lui *et al.* [LU98] studies infinite capacity fork-join queues with homogeneous servers. Each server has a *k-Erlang* distribution with identical mean service time. The goal of the paper is to present computationally feasible approximations. Lui achieves this approximation by noting that the state space is not uniformly distributed. For servers with identical service time, the state space is most likely to be clustered around an {n,n,n…} state space, where all queues have the same number of jobs waiting. His approximation is to limit the radius of calculation. One might, for instance, only consider state spaces ±5 tasks. Lui imposes the restraint that if a task departure would violate the maximum distance between queues, it cannot depart and must repeat the last stage of service. We will use an idea similar to this in our model. We will intuitively argue in our approximation that if both the fastest server and the slowest server have no jobs waiting, then neither should any intermediate servers.

Section 3: Problem Model

Our state space is restricted to three dimensions: $\{n_1, n_k, n_0\}$, as per Figure 1. We assume that server 1 is the fastest system (largest μ) and server k is the slowest system (smallest μ). We estimate the number of jobs in the system as $M = \frac{n_1 + n_k + n_o}{k}$. We claim that the set of state transitions are as shown below. The numbers in the right hand column refer to *state transition equations*.

n_1, n_k, n_o	n_1+1, n_k+1, n_o	$\lambda, M < N, n_1 < n, n_k < N$	(1)
	n_1-1, n_k, n_o+1	$\mu_1, n_1 > 0$	(2)
	n_1, n_k-1, n_o+1	$\mu_k, n_k > 0$	(3)
	n_1, n_k, n_o+1	$\frac{1}{k-2}\sum_{i=2}^{k-1}\mu_i p_i,\ M < N$	(4)
	n_1, n_k, n_o-k	$N/k\Delta,\ n_0 \geq k$	(5)
0,0,0	1,1,0	λ	

The fourth and fifth transitions merit discussion. Transition (4) represents a transition from an intermediate server (not the fastest nor the slowest). P_i is the probability that server i is busy given the current system state. We shall call this the *Intermediate Server Probability*. This is our approximation to reduce the state space. Note that we use an "average" transition rate, since we divide by *k-2*. Transition (5) reflects the periodic drain on the model. We call the term $N/(k\Delta)$ the *drain rate*. Since we generate up to k tasks in the join queue for each job, we drain the join queue at a rate scaled by the number of tasks we generate. The factor N enters the equation since in the problem all joined tasks leave the system simultaneously. This is a crude estimate. We find that this transition rate plays a critical role in our model.

We define a renewal period as when all server queues (excluding the join queue) are empty. We need to count the total number of jobs that pass through server *1* and server k since the beginning of each renewal period. We call this number $N_{i,\tau}$, where i is the server and τ is the renewal period. This is predominately for expository purposes. We do not maintain such a state in the model. We will later show an approximation for $N_{i,\tau}$ below.

For transition (4), we model a series of completion times at each server. Each completion time is an exponential random variable. A series of *m* completions is an *m-Erlang* random variable. To compute the Intermediate Server Probability, we consider three cases. If the fastest server is busy, we compute the probability that an intermediate server *i* has completed *N* tasks in the same or less time than server *1* has completed only $N_{1,\tau}$ tasks. If the fastest server is empty, but the slowest server is busy, we compute the probability that an intermediate server *i* completed all M tasks in the same or less time than server *k* completed $N_{k,\tau}$ tasks. If both the fastest and slowest servers are idle, we state that p_i is zero. We cannot compute similar probabilities when there are no tasks in the queues, since we do not know how long a time has elapsed since the queues were emptied. When we have jobs waiting, we know the time is bounded. They are bounded, because the service queues are *not* independent, but all share equal arrivals.

Figure 3 shows an example of the ISP. There are *M=5* jobs in the system. Server 1 has processed three jobs. Server *k* has processed one job. In this example, we would compute the Intermediate Server Probability based on server 1 (case 1 below). If n_1 were empty, we would use server *k* (case 2 below).

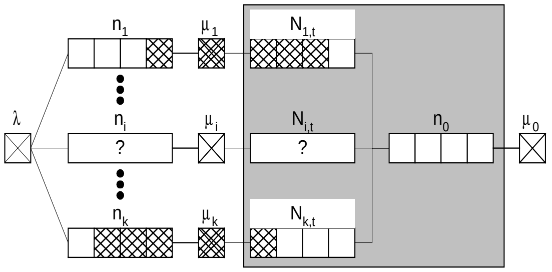


Figure 3. Bounding Processing Times, M=5.
Shaded area shows Join Queue.

Case 1: $n_1 > 0$

We consider the case that the fastest server has n_1 tasks. We wish to compute the probability that some intermediate server *i* has $n_i > 0$, given n_1. We would equate this to one minus the probability that server *i* has completed $n_1 + N_{1,\tau}$ tasks at least as fast as server *1* has completed $N_{1\ \tau}$ tasks. The probability that server *1* completes $N_{1,\tau}$ in time *t* is given by the Erlang distribution

$$f_1\ (t \mid k = N_{1,\tau}) = \frac{\mu_1(\mu_1 t)^{k-1}}{(k-1)!} e^{-\mu_1 t}$$

The probability that server i completes $n_1 + N_{1,\tau}$ tasks in time $T \leq t$ is given by the Erlang cumulative distribution

$$F_i(t \mid j = n_1 + N_{1,\tau}) = 1 - \sum_{u=0}^{j-1} \frac{(\mu_i t)^u}{u!} e^{-\mu_i t}$$

We may combine these equations to find the probability that server i completes all $n_1 + N_{1,\tau}$ quicker than server 1 may complete $N_{1,\tau}$ tasks as shown. We take the definitions of j and k from the above equations.

Eq 1
$$P[n_i = 0 \mid j,k] = \int_0^{\infty} \left(\left[1 - \sum_{u=0}^{j-1} \frac{(\mu_i t)^u}{u!} e^{-\mu_i t} \right] \left[\frac{\mu_1 (\mu_1 t)^{k-1}}{(k-1)!} e^{-\mu_1 t} \right] \right) dt$$

Since we model a continuous time Markov chain, we integrate over all possible times that it takes server 1 to complete k tasks. We multiply the cdf $F(t|j,k)$ times the probability that it took server 1 t time units to complete j tasks. The probability that we seek is found as $P[n_i>0] = 1-P[n_i=0]$.

Although the integral is very difficult to solve analytically, it yields to numerical methods. Following *Numerical Recipes* [PR88], we may solve the "improper" integral as a Romberg function on an open interval.

Case 2: $n_1 = 0$ and $n_k > 0$

This will be similar to Case 1. We wish to find the probability that server i took longer to process $n_k + N_{k,\tau}$ tasks than server k took to process $N_{k,\tau}$ tasks. The equations follow the same form as Case 1.

Case 3: $n_1 = 0$ and $n_k = 0$

In this case, we have no marking in the system by which to measure time. There could be any finite interval since servers *1* and *k* finished processing their tasks. In this case, we must resort to the argument that since the slowest server and the fastest server have finished processing all tasks, it is likely that all intermediate servers have also finished their tasks. We also note our argument above that we cannot bound time in this case, as we can when there are tasks waiting.

The "sender" of the jobs knows several global state variables. N is the maximum number of allowable jobs in the system, J is the number of jobs removed from the system. The number of jobs removed from the system would be $J = min(M – max(n_1, n_k), n_0 / k)$. One should not confuse these "evaluation" epochs with the renewal process mentioned above. The renewal processes generate the markers $\{N_{i,\tau}\}$. Our state is the 3-tuple $\{n_1, n_k, n_0\}$.

We create an infinitesimal transition rate matrix Q from the above equations. Using an appropriate numerical method, we solve the eigenvalue equation $\boldsymbol{\pi}\ Q = 0$, where $\boldsymbol{\pi}$ is the steady-state probability matrix.

To have any chance at writing the infinitesimal generator, we need to simplify the equations for p_i. We will approximate $N_{i,\tau}$, being the number of tasks processed at server i since the last time we were in state {0,0,0}, by $N_{i,\tau} = max(M – n_i, 1)$. This will make our renewal process coincide with the evaluation epochs. Now, all variables only depend on the current state and the global variable N.

Section 4: Model Analysis

We discuss several aspects of our model. We begin with a discussion of the effectiveness of our model at limiting the state space. Based on our experience with the model, we comment on the robustness and stability of the Markov chain.

One goal of our model was to reduce the state space to a manageable size. By limiting the system to three queues regardless of the number of servers involved, we hoped to keep the state space small. As it turns out, since the domain of n_0 is *[0…kN]*, we still have dependence on the number of servers. A three server model with *N=10* will generate 2541 states. A 2541x2541 *Q* matrix may still be solved with simple methods, such as LU decomposition with equation substitution [PR88, ST94], if one has at least 60 MB of memory. For *N=10*, there are 4961 states for *k=5* and 6171 for *k=6*. One may approximate the number of states as $S \approx {}^{1}/_{3}\, kN^{3}$. We observed the 1/3 scaling factor from our various trails. The maximum number of array elements will be approximately 5 times the number of states. A system with *k=5* and *N=50* would have approximately 625,000 array elements. While these numbers are much better than a model with all *k* servers, they still require sparse storage and iterative methods [e.g. ST94].

We found that our model is very sensitive to the drain rate in transition equation (5). The problem description states that with periodicity Δ, the system purges all joined jobs. In a Markov model, we must approximate the purge process. We chose a transition rate that scales with the number of jobs in the system. It reduces the join queue by *k* tasks for each purge event. We experimented with other transition rates. We tried, for instance using our approximate of *M* in place of *N*. We also looked at reducing the state to $n_0 = n_0 \bmod k$ with a slower rate. We found that these formulae were less accurate than the simple equation we used.

Since the model is very sensitive to the drain rate, and it is not obvious what value one should use, we are not confident that the model exhibits stable behavior. Modest changes in the draining rate may cause large swings in the output. For this reason, we do not believe this model would work well outside of certain parameters.

We found from our results that the model performed poorly when the drain rate $N/(k\,\Delta) \approx 1/\mu_{max}$. This represents an "equilibrium" state where there are probably about as many jobs waiting for service as there are jobs waiting in the join queue. In this context, the model is hypersensitive to transitions from the service facilities, which is where we make a large approximation.

In our implementation of the model, we used a simple LU decomposition with equation substitution. We followed the LU decomposition algorithm from *Numerical Recipes* [PR88]. The matrices were generated following [ST94]. This method transposes the Q matrix and replaces the last row with 1's. Letting Z equal the modified, Q matrix, one solves $\boldsymbol{\pi} Z = \boldsymbol{e}$, where $\boldsymbol{e} = (0,0,\ldots,0,1)^T$. We found that the LU method of [PR88] solved this non-homogeneous system better than the original $\boldsymbol{\pi}\ Q = 0$, for which it always returned the trivial **0** matrix.

Section 5: Simulation

We used the *Sim++* simulation environment from the *simpack3* distribution [FI95]. Below, we discuss our simulation design and the methods we used to track system parameters. We describe the statistics that the simulation produced. A job refers to an arrival before splitting in to parallel tasks. A task is *1/k* parts of a job, where *k* is the number of servers.

Simpack uses **tokens**, which pass through **facilities**. There is a universal **future**, which schedules up coming events. A scheduled event has a time and C++ callback function. A token may contain several attributes. These attributes may carry information through the simulation.

Our simulation begins by calling a "kickoff" function that schedules 10,000 job arrivals from a Poisson distribution. Each job is assigned a Global ID (GID). A job generates *k* task tokens. Each task token has a Unique ID (UID) and GID. A task token is scheduled for a specific facility. There are *k* facility objects, each with a unique facility ID. The token also stores the facility ID of the queuing facility. Storing the facility ID in the token allows us to use only one instantiation of each facility function. The facility functions may lookup the facility ID from the token and index to an array of pointers of the facility class.

Facilities for service queues are modeled with three functions. We model the join queue with a different set of functions. The **arrival()** function checks the system to see if the system is blocked. It does this by looking at the maximum number of jobs in each facility queue and the number of jobs in the join queue. If the system is blocked, it will discard the arrival. Otherwise, the arriving token is scheduled to the **request()** function with zero delay. If the server is available, **request()** schedules the **release()** function some time in the future based on an exponential distribution and the facility's service rate. With zero delay, the **release()** function schedules the token to the join facility arrival function, **S0_arrival()**.

The join facility also has three functions. The function **S0_arrival()** accepts a token from a facility's **release()** function. **S0_arrival()** uses an associative cache to count the arrival of tokens with the same GID. We schedule each token for the function **S0_purge()** at the next purge time. The function **S0_purge()** examines the associative cache. Any GID that has *k* arrivals is freed and appropriate counters incremented. The third function periodically collects statistics on the purge process.

We store a two-dimensional array, **pmf[jobs][removed]**. The *jobs* dimension is the number of jobs currently in the system. The *removed* dimension is the number of GID's **S0_purge()** released since the last purge event. We schedule a function **S0_stats()** just after **S0_purge()**. The stats function examines the number of GID's released and updates the pmf. It also updates the *NextPurge* variable, which stores the time of the next **S0_purge()** event. *NextPurge* is either constant or exponential, depending on startup parameters. If there are any tokens still in the system, **S0_stats()** schedules itself again.

The simulation will end when there are no more tokens in the system and the last **S0_stats()** event finishes. We may then compute statistics from the pmf array. We compute the expected number of jobs removed for each *jobs* index, where the *jobs* index is the number of jobs in the system. This yields an observed estimation of the probability *P[J | M],* where *J* is the number of jobs removed and *M* is the number of jobs in the system.

For *k=2* and *k=3* ran the simulation for ten repetitions with the same system parameters. For *k=5*, we ran for ten repetitions, but with only 5,000 arrivals. Each run reported *E[J | M]*, the expected number of jobs removed given the number of jobs in the system. We averaged this number of the ten repetitions. Each trial also reported the overall *E[J]*, which we use to compare results with the analytical model.

Section 6: Simulation and Model Comparison

We found that our approximation model generally agrees with simulation results. Figures 4 to 21 show the probability mass function for *E[J | M]* from our simulation runs. We also show the 95% confidence interval for each simulation result. A confidence interval of exactly 0 indicates the variance was exactly zero, which usually means that the value only appeared once. Each data series also includes a valued labeled "all". This represents *E[J]* over all *M*. We compare *E[J]* with the results from our model. The data point for the model is labeled "Model." The confidence value for the Model is actually the percentage difference between the "Model" value and the "All" value. We compute the percentage difference as *(Model – All) / Model*.

Each graph has a specific *k* value, which is the number of "receiver" queues. It is the number of parallel systems. *N* is the maximum number of jobs allowed in the system. The *Skew* parameter is the amount of change between each server. The "fastest" server has the stated service period ($1/\mu$). Each server beyond the first has a service period of $1/\mu + k * skew$. The other values have standard meanings, as defined earlier in the paper.

In general, our model was within 10%, except for the $k=3$, $N=10$, $\Delta=5$ series (Figures 4-7). This series was consistently 30% over simulation. We would expect that it has to do with how we estimated the drain rate in the Markov model. For the same parameters, but with $\Delta=10$ (Figures 8-11), we find the model is in reasonable agreement with simulation.

We also included two data series with $k=5$ servers. We only run these for $N=5$, which generated 756 states. Our model performed well against the simulation results. The simulations were only run for 5,000 arrivals, as opposed to the 10,000 arrivals for other data series as a time saving measure. The model was within 3% of the simulation results.

We see from figures for the *k=2* series (Figures 12-19) that the modeling and the simulation agree reasonably well. Half the series are under 5% difference, and all are under 14%. The series that are over 5% difference are those where the drain rate is comparable to the service rate.

In the *k=3* series (Figures 4-11), we see similar behavior. In Figures 4 to 7, the model performs poorly compared to the simulation. The results are consistently about 30% over simulation. In these cases, the drain rate is too small for the number of jobs generated by the Markov model.

The *k=5* series (Figures 20-21) agree very well with simulation. Our results are within 3% for the two series we analyzed.

Section 7: Conclusion

The general agreement of our model with simulation, usually within 10%, shows promise that our intermediate server approximation could model large *k* systems. An implementation that uses sparse matrices and iterative methods would be required for larger *N* values. A key point of further research is how to better model the draining from the Markov chain. Our approximation for the purge process is not always suitable.

We note that when the drain rate is approximately equal to the service rate, our model does not match simulation well. We believe this is an effect of our Markov model. Our approximation for *M*, the number of jobs in the system, does not account for tasks in the intermediate queues. When the drain rate is approximately equal to the service rate, the number of unaccounted tasks in the intermediate queues is at its peak. This is also the case when a large number of jobs come into the join queue via the Intermediate Server Probability . One may be able to refine our model by a better accounting of these deficits.

References

[BA98] Balsamo, Simonetta, L. Donatiello, N. Van Dijk, "Bound Performance Models of Heterogeneous Parallel Processing Systems," IEEE Trans. PDS, v. 9 n. 10, Oct. 1998, pp. 1041-1056.

[DA94] Dallery, Yves, Z. Liu, D. Towsley, "Equivalence, Reversibility, Symmetry and Concavity Properties in Fork-Join Queuing Networks with Blocking," J. ACM, v. 41 n. 5, Sept. 1994, pp. 903-942.

[DA97] Dallery, Yves, Z. Liu, D. Towsley, "Properties of Fork/Join Queuing Networks with Blocking Under Various Operating Mechanisms," IEEE Trans. Robotics and Automation, v. 13 n. 4, Aug. 1997, pp. 503-518.

[FI95] Fishwick, Paul A., *Simulation Model Design and Execution. Building Digital Worlds.* Prentice-Hall, Inc., New Jersey, 1995.

[FL84] Flatto, L, S. Hahn, "Two Parallel Queues Created by Arrivals With Two Demands I", SIAM J. Applied Math, v. 44, n. 5, Oct 1984, pp. 1041-1053.

[FL85] Flatto, L, S. Hahn, "Two Parallel Queues Created by Arrivals With Two Demands II", SIAM J. Applied Math, v. 45, n. 5, Oct 1985, pp. 861-878.

[FR97] Frostig, Esther, T. Lehtonen, "Stochastic Comparisons for Fork-Join Queues with Exponential Processing Times," J. Appl. Prob, v. 34, 1997, pp. 487-497.

[LU98] Lui, John C. S., R. Muntz, D. Towsley, "Computing Performance Bounds of Fork-Join Parallel Programs Under a Multiprocessing Environment," IEEE Trans. PDS, v. 9 n. 3, Mar. 1998, pp. 295-311.

[NE81] Neuts, Marcel. F., *Matrix-Geometric Solutions in Stochastic Models. An Algorithmic Approach*, Dover Publications, Inc., New York: 1981.

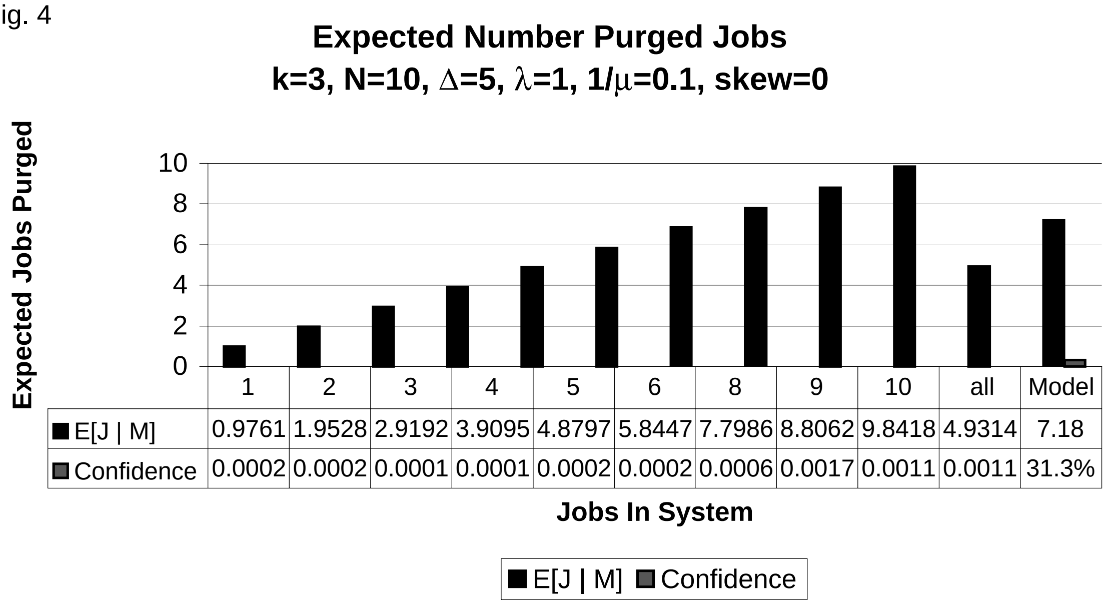
[PR88] Press, William H., B. Flannery, S. Teukolsky, W. Vetterling, *Numerical Recipes in C. The Art of Scientific Computing*, Cambridge University Press, Cambridge: 1988.

[ST94] Stewart, William J., *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press: New Jersey, 1994.

[TH94] Thomasian, Alexander, A. Tantawi, "Approximate Solutions for M/G/1 Fork/Join Synchronization," Proc. 1994 Winder Simulation Conference, Florida Dec. 11-14, 1994, pp. 361-368.

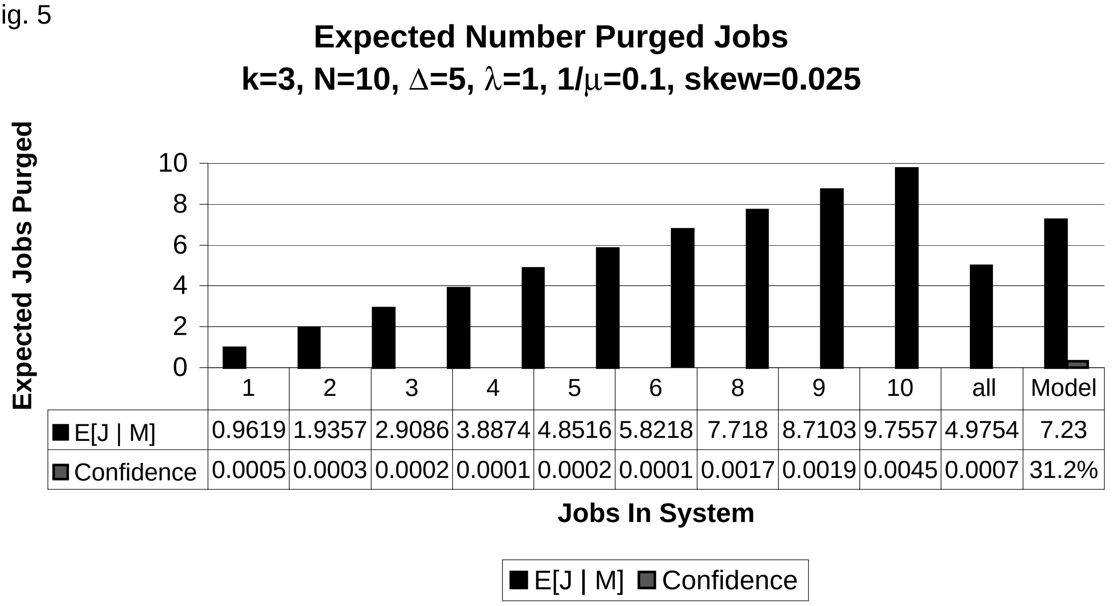
[VA96] Varki, Elizabeth, L. Dowdy, "Analysis of Balanced Fork-Join Queuing Networks," Perf. Eval. Rev., v. 24 n. 1, May 1996, pp. 232-241.

Fig. 4



	1	2	3	4	5	6	8	9	10	all	Model
■ E[J | M]	0.9761	1.9528	2.9192	3.9095	4.8797	5.8447	7.7986	8.8062	9.8418	4.9314	7.18
▣ Confidence	0.0002	0.0002	0.0001	0.0001	0.0002	0.0002	0.0006	0.0017	0.0011	0.0011	31.3%

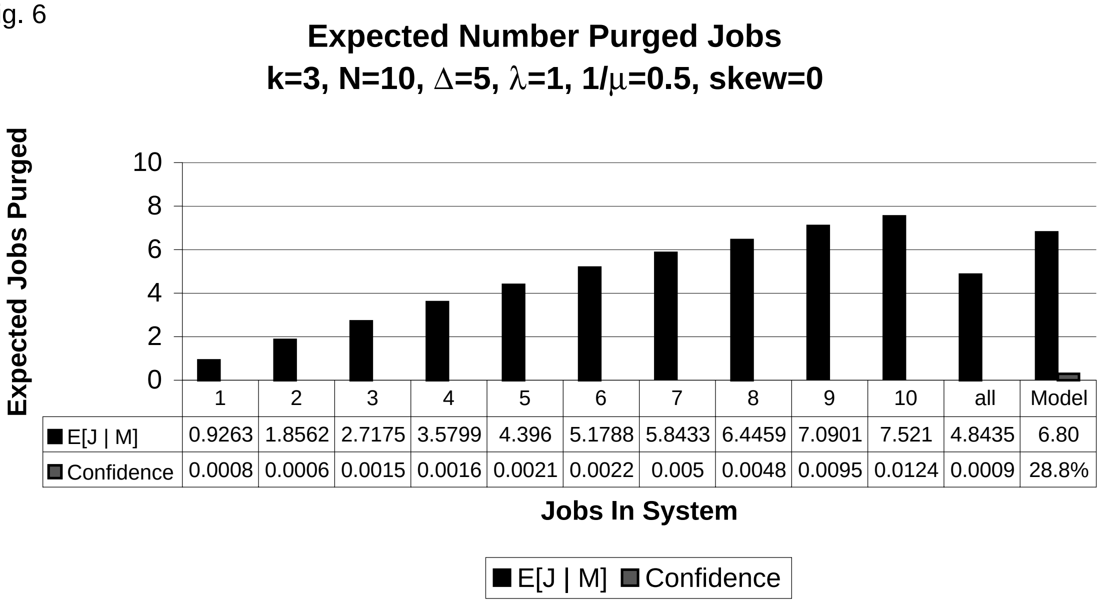
Fig. 5



	1	2	3	4	5	6	8	9	10	all	Model
■ E[J | M]	0.9619	1.9357	2.9086	3.8874	4.8516	5.8218	7.718	8.7103	9.7557	4.9754	7.23
▣ Confidence	0.0005	0.0003	0.0002	0.0001	0.0002	0.0001	0.0017	0.0019	0.0045	0.0007	31.2%

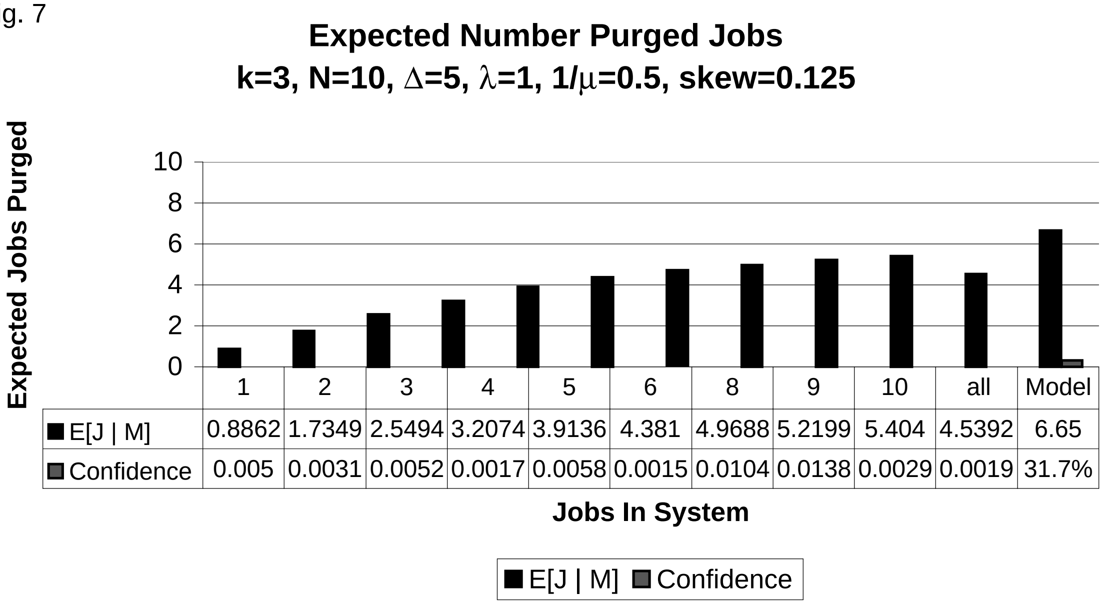
k = # queues
N = max jobs
Δ = Purge interval
λ = arrival rate
1/μ = Fastest server period
skew = 1/μ increment per server

Fig. 6



	1	2	3	4	5	6	7	8	9	10	all	Model
■ E[J | M]	0.9263	1.8562	2.7175	3.5799	4.396	5.1788	5.8433	6.4459	7.0901	7.521	4.8435	6.80
▨ Confidence	0.0008	0.0006	0.0015	0.0016	0.0021	0.0022	0.005	0.0048	0.0095	0.0124	0.0009	28.8%

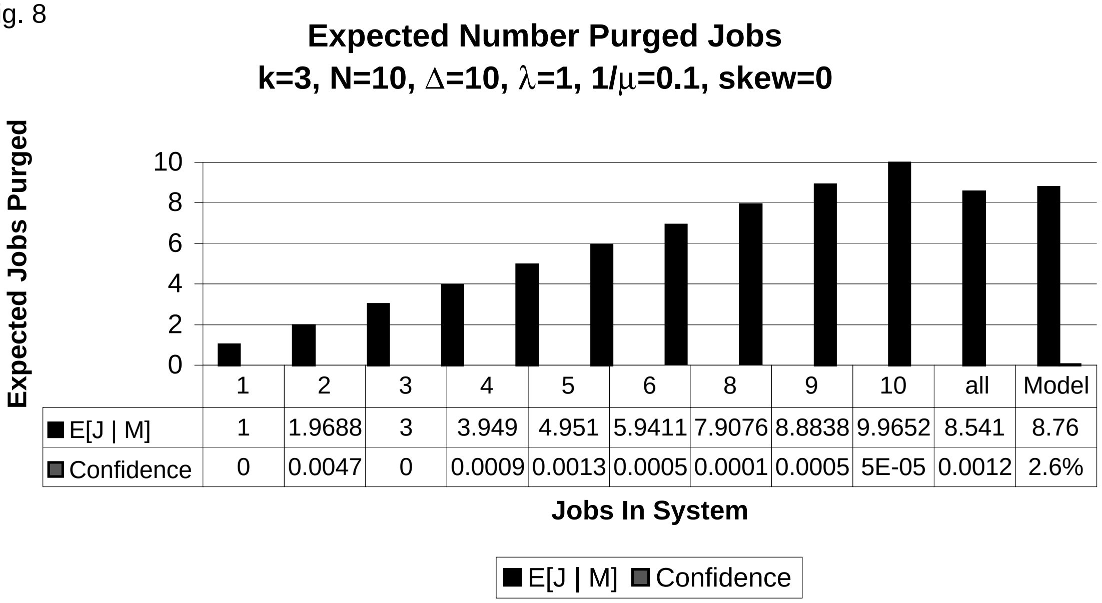
Fig. 7



	1	2	3	4	5	6	8	9	10	all	Model
■ E[J | M]	0.8862	1.7349	2.5494	3.2074	3.9136	4.381	4.9688	5.2199	5.404	4.5392	6.65
▨ Confidence	0.005	0.0031	0.0052	0.0017	0.0058	0.0015	0.0104	0.0138	0.0029	0.0019	31.7%

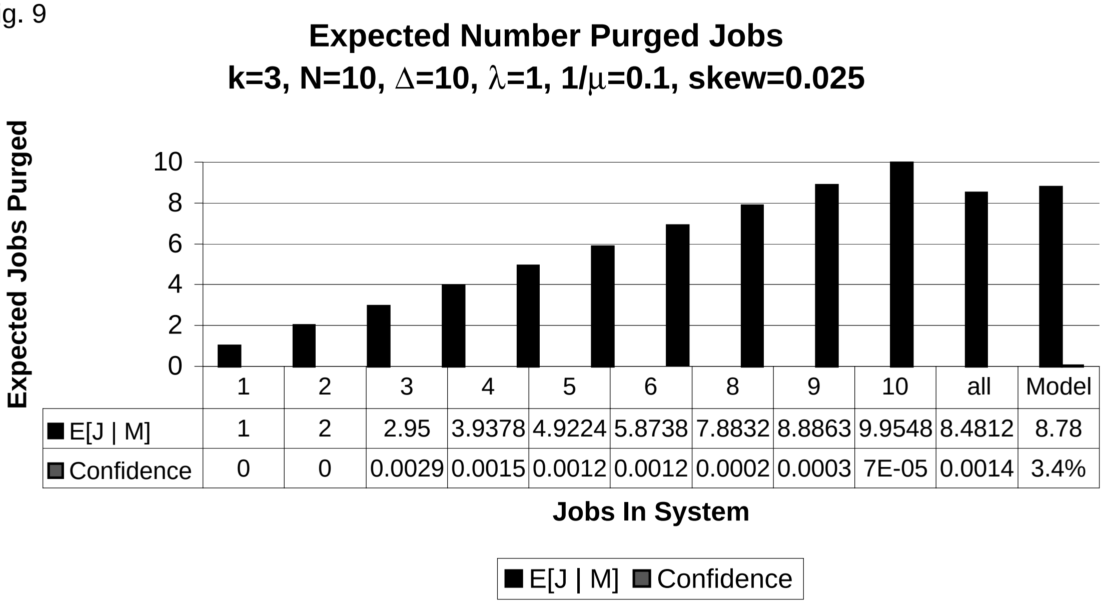
k = # queues
N = max jobs
Δ = Purge interval
λ = arrival rate
1/μ = Fastest server period
skew = 1/μ increment per server

Fig. 8



	1	2	3	4	5	6	8	9	10	all	Model
■ E[J | M]	1	1.9688	3	3.949	4.951	5.9411	7.9076	8.8838	9.9652	8.541	8.76
■ Confidence	0	0.0047	0	0.0009	0.0013	0.0005	0.0001	0.0005	5E-05	0.0012	2.6%

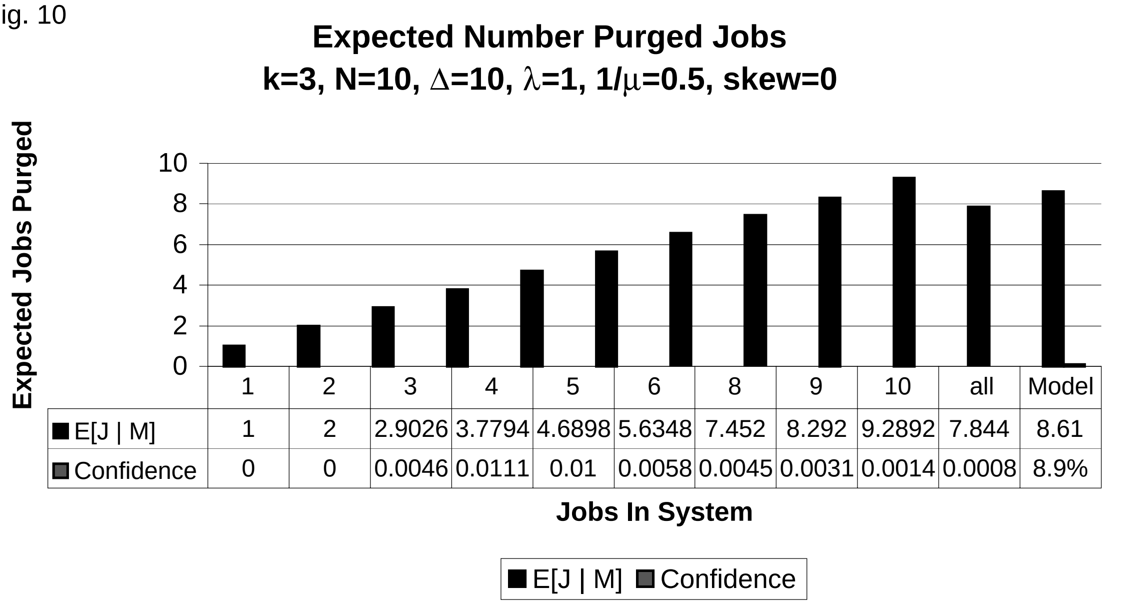
Fig. 9



	1	2	3	4	5	6	8	9	10	all	Model
■ E[J | M]	1	2	2.95	3.9378	4.9224	5.8738	7.8832	8.8863	9.9548	8.4812	8.78
■ Confidence	0	0	0.0029	0.0015	0.0012	0.0012	0.0002	0.0003	7E-05	0.0014	3.4%

k = # queues
N = max jobs
Δ = Purge interval
λ = arrival rate
1/μ = Fastest server period
skew = 1/μ increment per server

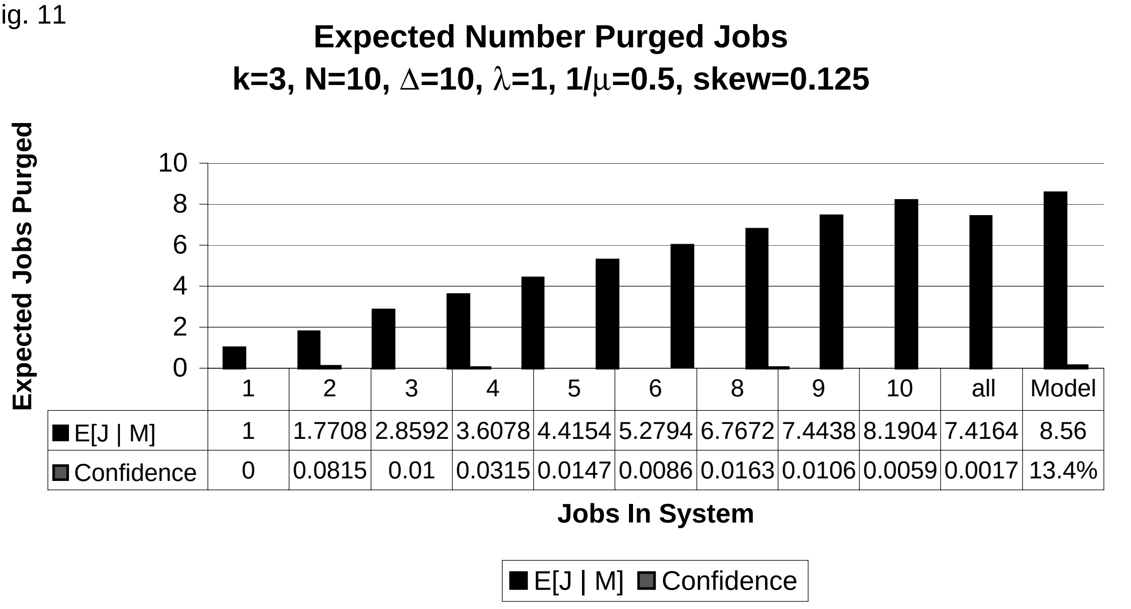
Fig. 10



Expected Number Purged Jobs
k=3, N=10, Δ=10, λ=1, 1/μ=0.5, skew=0

	1	2	3	4	5	6	8	9	10	all	Model
■ E[J | M]	1	2	2.9026	3.7794	4.6898	5.6348	7.452	8.292	9.2892	7.844	8.61
■ Confidence	0	0	0.0046	0.0111	0.01	0.0058	0.0045	0.0031	0.0014	0.0008	8.9%

Fig. 11

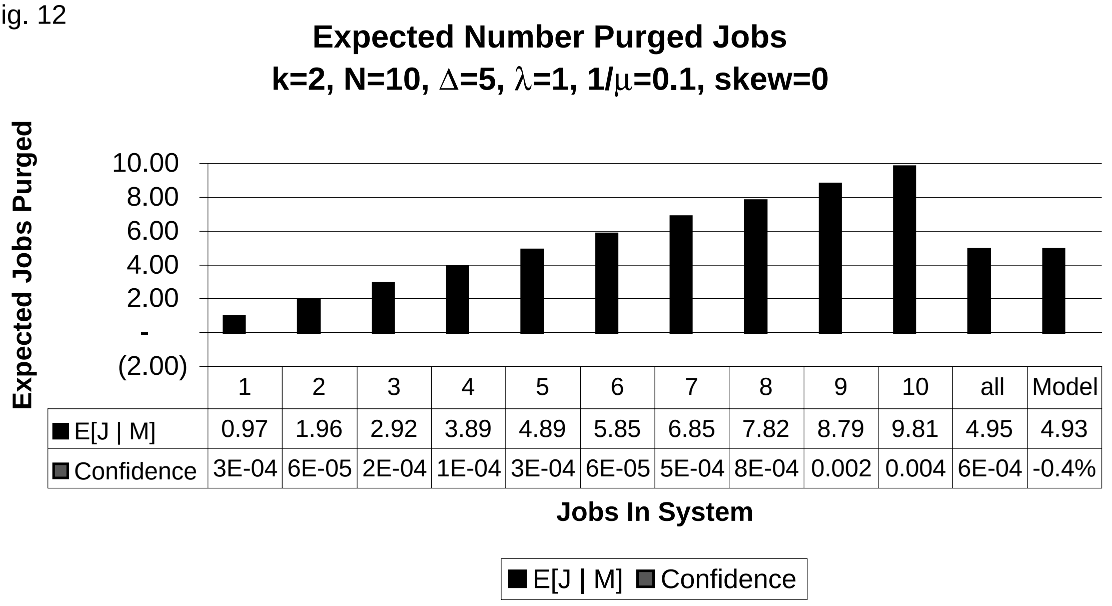


Expected Number Purged Jobs
k=3, N=10, Δ=10, λ=1, 1/μ=0.5, skew=0.125

	1	2	3	4	5	6	8	9	10	all	Model
■ E[J | M]	1	1.7708	2.8592	3.6078	4.4154	5.2794	6.7672	7.4438	8.1904	7.4164	8.56
■ Confidence	0	0.0815	0.01	0.0315	0.0147	0.0086	0.0163	0.0106	0.0059	0.0017	13.4%

k = # queues
N = max jobs
Δ = Purge interval
λ = arrival rate
1/μ = Fastest server period
skew = 1/μ increment per server

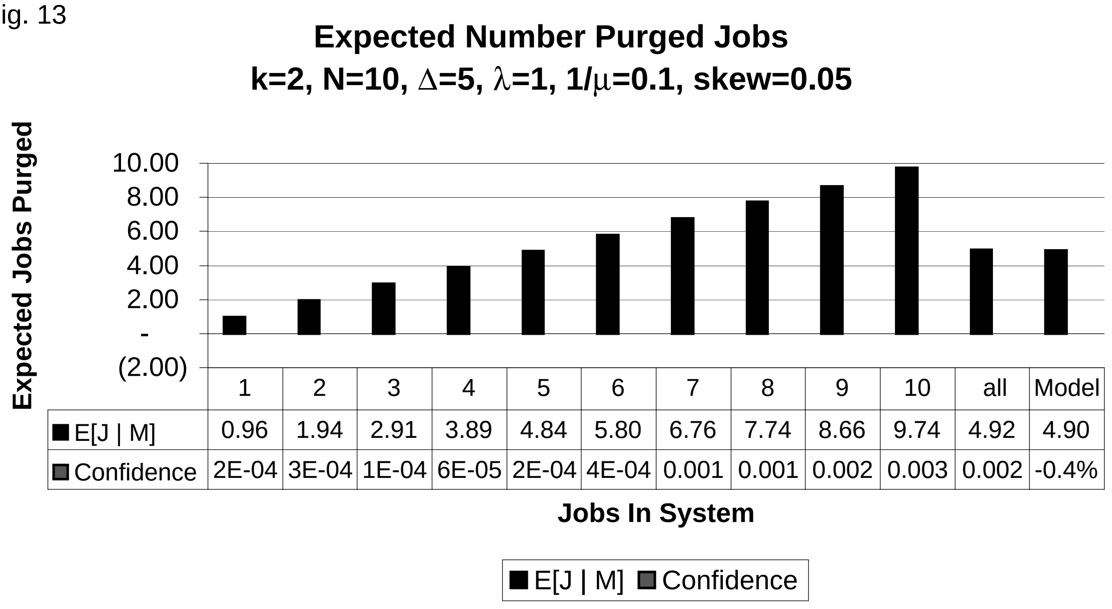
Fig. 12



Expected Number Purged Jobs
k=2, N=10, Δ=5, λ=1, 1/μ=0.1, skew=0

	1	2	3	4	5	6	7	8	9	10	all	Model
■ E[J | M]	0.97	1.96	2.92	3.89	4.89	5.85	6.85	7.82	8.79	9.81	4.95	4.93
▩ Confidence	3E-04	6E-05	2E-04	1E-04	3E-04	6E-05	5E-04	8E-04	0.002	0.004	6E-04	-0.4%

Fig. 13



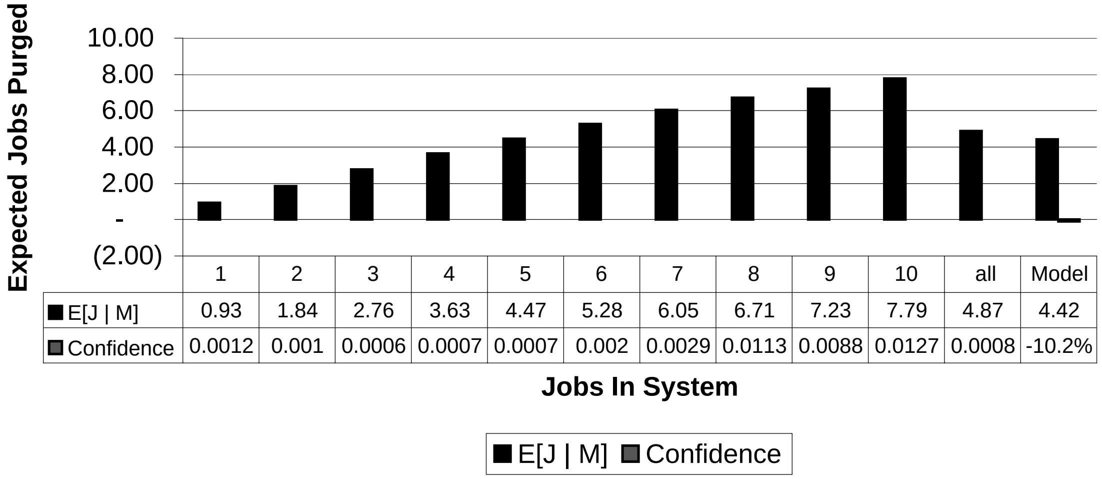
Expected Number Purged Jobs
k=2, N=10, Δ=5, λ=1, 1/μ=0.1, skew=0.05

	1	2	3	4	5	6	7	8	9	10	all	Model
■ E[J | M]	0.96	1.94	2.91	3.89	4.84	5.80	6.76	7.74	8.66	9.74	4.92	4.90
▩ Confidence	2E-04	3E-04	1E-04	6E-05	2E-04	4E-04	0.001	0.001	0.002	0.003	0.002	-0.4%

k = # queues
N = max jobs
Δ = Purge interval
λ = arrival rate
1/μ = Fastest server period
skew = 1/μ increment per server

Fig. 14

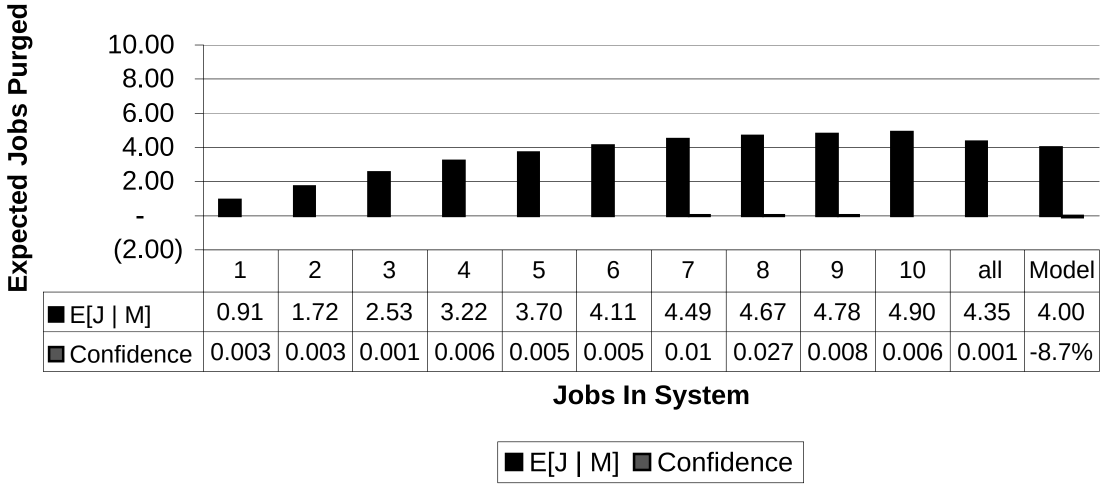
Expected Number Purged Jobs
k=2, N=10, Δ=5, λ=1, 1/μ=0.5, skew=0



	1	2	3	4	5	6	7	8	9	10	all	Model
■ E[J | M]	0.93	1.84	2.76	3.63	4.47	5.28	6.05	6.71	7.23	7.79	4.87	4.42
▣ Confidence	0.0012	0.001	0.0006	0.0007	0.0007	0.002	0.0029	0.0113	0.0088	0.0127	0.0008	-10.2%

Fig. 15

Expected Number Purged Jobs
k=2, N=10, Δ=5, λ=1, 1/μ=0.5, skew=0.25

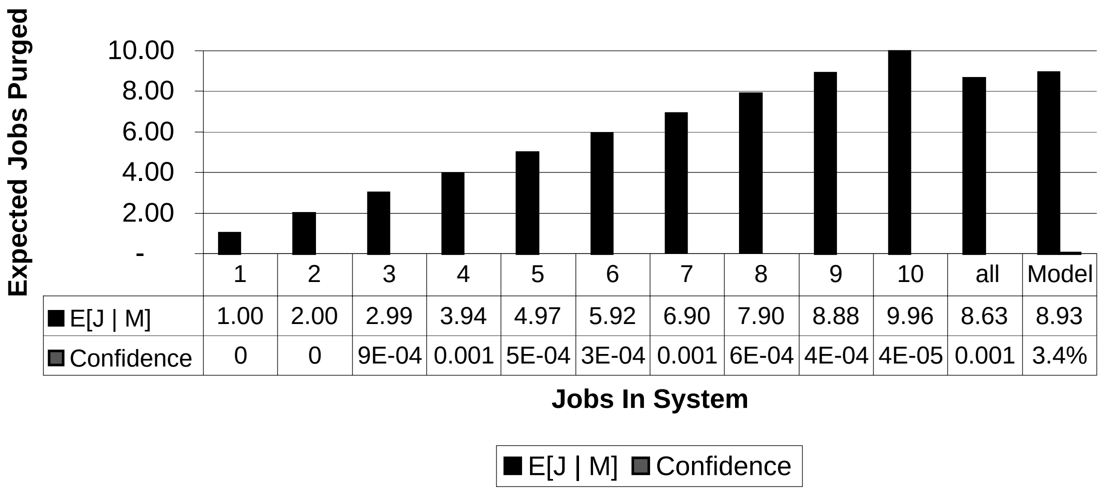


	1	2	3	4	5	6	7	8	9	10	all	Model
■ E[J | M]	0.91	1.72	2.53	3.22	3.70	4.11	4.49	4.67	4.78	4.90	4.35	4.00
▣ Confidence	0.003	0.003	0.001	0.006	0.005	0.005	0.01	0.027	0.008	0.006	0.001	-8.7%

k = # queues
N = max jobs
Δ = Purge interval
λ = arrival rate
1/μ = Fastest server period
skew = 1/μ increment per server

Fig. 16

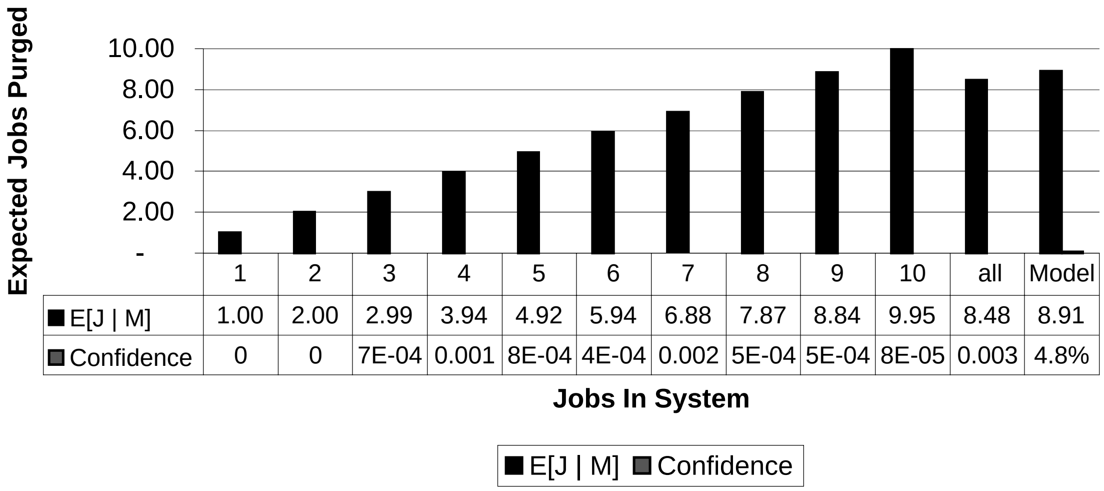
Expected Number Purged Jobs
k=2, N=10, Δ=10, λ=1, 1/μ=0.1, skew=0



	1	2	3	4	5	6	7	8	9	10	all	Model
■ E[J | M]	1.00	2.00	2.99	3.94	4.97	5.92	6.90	7.90	8.88	9.96	8.63	8.93
▨ Confidence	0	0	9E-04	0.001	5E-04	3E-04	0.001	6E-04	4E-04	4E-05	0.001	3.4%

Fig. 17

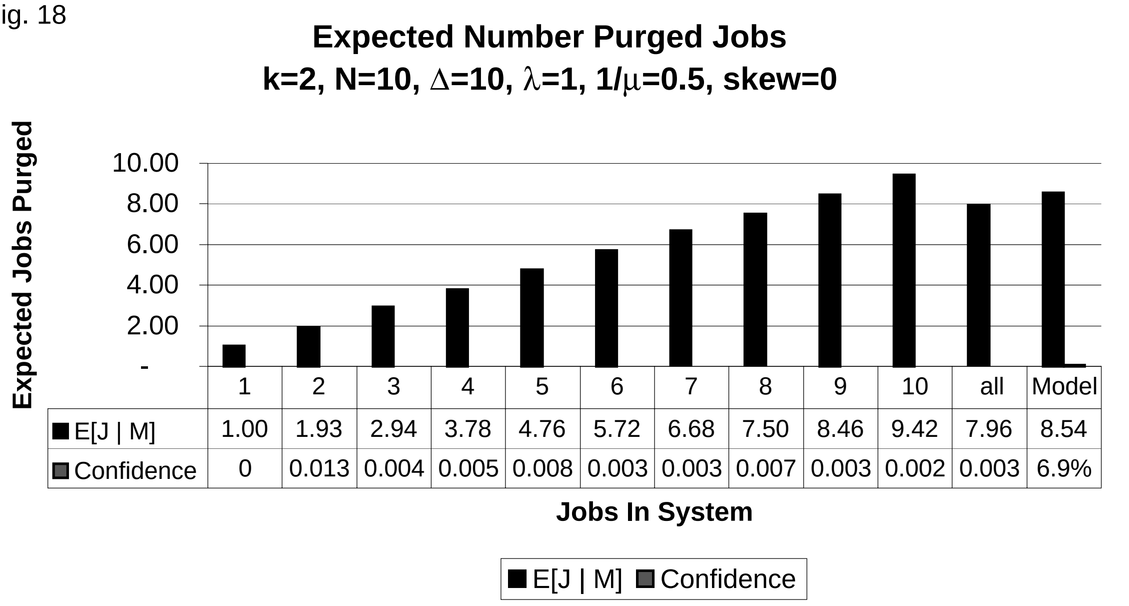
Expected Number Purged Jobs
k=2, N=10, Δ=10, λ=1, 1/μ=0.1, skew=0.05



	1	2	3	4	5	6	7	8	9	10	all	Model
■ E[J | M]	1.00	2.00	2.99	3.94	4.92	5.94	6.88	7.87	8.84	9.95	8.48	8.91
▨ Confidence	0	0	7E-04	0.001	8E-04	4E-04	0.002	5E-04	5E-04	8E-05	0.003	4.8%

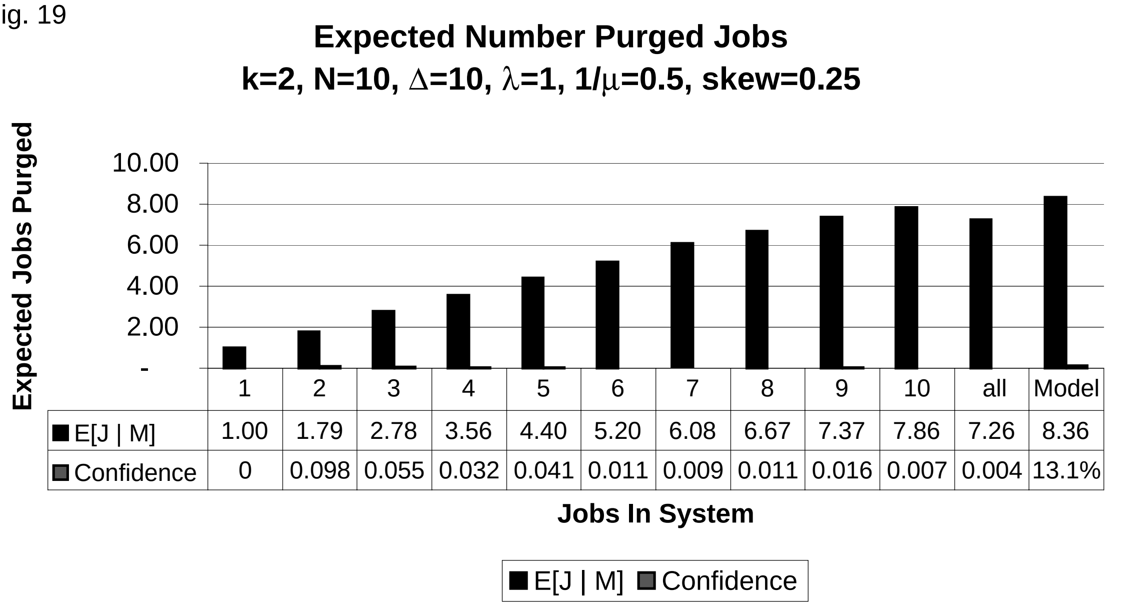
k = # queues
N = max jobs
Δ = Purge interval
λ = arrival rate
1/μ = Fastest server period
skew = 1/μ increment per server

Fig. 18



	1	2	3	4	5	6	7	8	9	10	all	Model
■ E[J | M]	1.00	1.93	2.94	3.78	4.76	5.72	6.68	7.50	8.46	9.42	7.96	8.54
■ Confidence	0	0.013	0.004	0.005	0.008	0.003	0.003	0.007	0.003	0.002	0.003	6.9%

Fig. 19

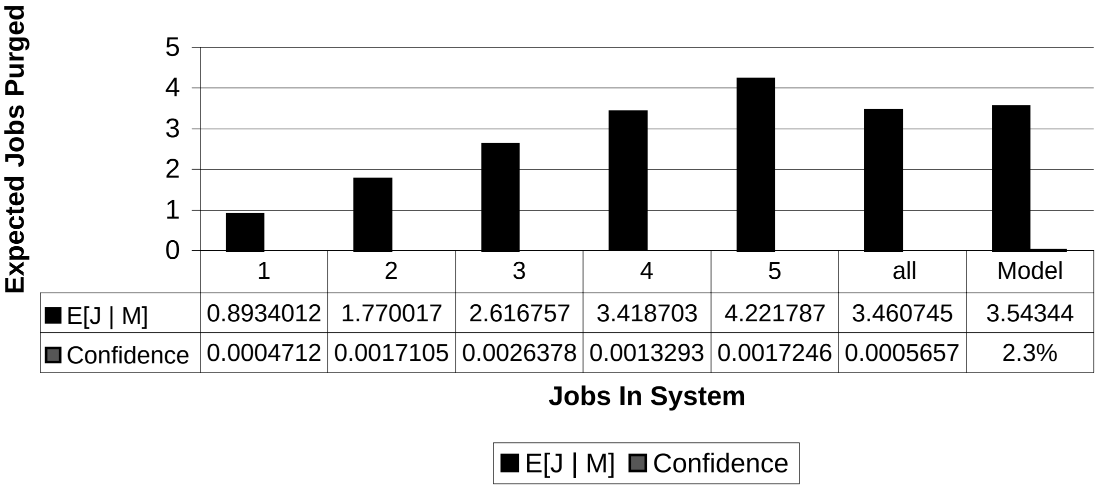


	1	2	3	4	5	6	7	8	9	10	all	Model
■ E[J | M]	1.00	1.79	2.78	3.56	4.40	5.20	6.08	6.67	7.37	7.86	7.26	8.36
■ Confidence	0	0.098	0.055	0.032	0.041	0.011	0.009	0.011	0.016	0.007	0.004	13.1%

k = # queues
N = max jobs
Δ = Purge interval
λ = arrival rate
1/μ = Fastest server period
skew = 1/μ increment per server

Fig. 20

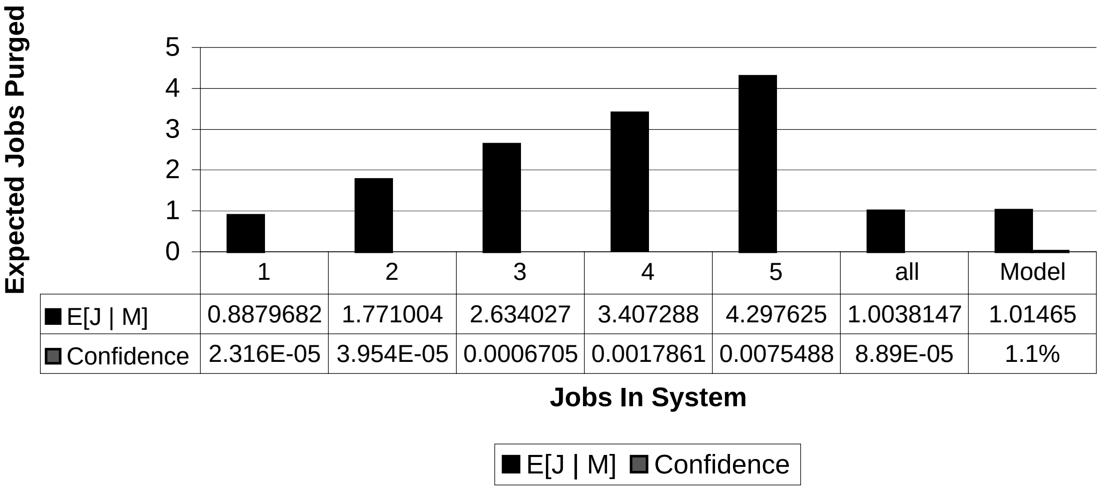
Expected Number Purged Jobs
k=5, N=5, Δ=5, λ=1, 1/μ=0.5, skew=0



	1	2	3	4	5	all	Model
■ E[J | M]	0.8934012	1.770017	2.616757	3.418703	4.221787	3.460745	3.54344
■ Confidence	0.0004712	0.0017105	0.0026378	0.0013293	0.0017246	0.0005657	2.3%

Fig. 21

Expected Number Purged Jobs
k=5, N=5, Δ=1, λ=1, 1/μ=0.1, skew=0



	1	2	3	4	5	all	Model
■ E[J | M]	0.8879682	1.771004	2.634027	3.407288	4.297625	1.0038147	1.01465
■ Confidence	2.316E-05	3.954E-05	0.0006705	0.0017861	0.0075488	8.89E-05	1.1%

k = # queues
N = max jobs
Δ = Purge interval
λ = arrival rate
1/μ = Fastest server period
skew = 1/μ increment per server