PERFORMANCE OF STOCHASTIC RENDEZVOUS NETWORKS WITH PRIORITY TASKS

J.W. Miernik, C.M. Woodside, J.E. Neilson and D.C. Petriu
Department of Systems and Computer Engineering
School of Computer Science
Carleton University, Ottawa, Canada K1S 5B6

February 18, 1998

Abstract

Multi-tasking multiprocessor systems frequently use rendezvous for intertask communication, and Stochastic Rendezvous Networks are a form of performance model for such systems. Tasks are frequently given different priorities to adjust the system performance. This paper considers a pre-emptive-resume priority discipline for executing tasks sharing each processor. An iterative algorithm is proposed, based on the well-known "MVA Priority Approximation". Throughput errors below 10% are found in many cases. The worst accuracy (as usual) is for low-priority tasks on highly-utilized processors.

1 Introduction

This paper considers the performance of multiprocessor multi-tasking systems with rendezvous, and the effect of priorities among tasks. Multi-tasking software applications are growing in importance, either for single processors, or for multi-processors and networks. Performance models of these systems should take into account the software communications behavior, which introduces queueing and service delays and may cause software bottlenecks. This is particularly true of software which uses synchronous, or rendezvous (RNV) communication, in which a sender of a message must wait for it to be received (and in some systems, for a reply). Systems written in the Ada language [DOD83] (in which messaging has a call/return semantics), or running under the XMS [GAMM84] or V [CHER83] distributed operating systems, use RNV communications. The advantage obtained with this programming style is synchronized task behavior, but there may well be a performance penalty due to task waiting.

There are three approaches to modelling the performance of multi-tasking software: with queueing networks, with detailed Petri Nets or Markov chains, or with an intermediate-level model like Rendezvous Networks. In the queueing network approach the existence of task communication is suppressed and contention for processors is described by a queueing model, with additional components to approximate the load due to parallel execution, and the synchronization delays. Heidelberger and Trivedi describe this approach in [HEID83]. A single processor with tasks and priorities was considered by Daigle [DAIG81]. The software was however limited to a linear sequence of tasks to be performed for each response, and to asynchronous communication (no RNVs). The model can be used to choose optimal priorities between tasks, to minimize response time.

In the Timed Petri Net approach a very detailed description of task behavior can be made (also, of hardware behavior). Stochastic execution times are modelled by the GSPN approach of Ajmone Marsan and Balbo [MARS84], and deterministic times by the GTPN approach of Holliday and Vernon [HOLL87]. A Markov chain model results in either case, and other Markov models have also been proposed (e.g. Radia [RADI84]). The problem of state explosion severely limits the size of systems that can be solved; in our experience the limit is well under a dozen tasks. Tasking models with Petri Nets have also been investigated, for example by Shatz [SHAT88] (although without time) and Kant [KANT87].

The Stochastic Rendezvous Nets (SRVN) approach [WOOD86], [MIER88] takes a middle road, fully representing some major features of tasks with rendezvous while suppressing or approximating other features. The SRVN waiting-time calculation analyzes a queue with the "two-phase" service given by RNV server tasks, where the first phase is executed between receving a message and generating a reply, and the second phase is executed after the reply. Both phases may include nested delays where the server tasks send to another server in the system. Because of these nested waiting delays a task may be a bottleneck even though it is mostly inactive and waiting. SRVNs generalize queueing networks in two ways – the nested service in which the originating server remains blocked, and the second phase of service. A normal server (such as a processor) is modelled as a RNV server with no nested RNVs and no second phase. Thus hardware queueing and software queueing can in principle be treated uniformly.

Figures 1 and 2 show an online database application. The software is composed of three tasks: one file server (task 3) and two channel tasks (tasks 1 and 2) serving data base access requests originating on two different priority communication channels. The channel tasks are allocated to processor 1, the file server is running on processor 2. A stochastic Petri net for the example is given in Figure 1 and the SRVN in Figure 2. We are interested in the evaluation of maximum throughput of the system, with backlogs of waiting requests from each channel. Thus we assume tasks 1 and 2 cycle as rapidly as possible, with no delay waiting for input.

Preemptive-resume priority service discipline is analysed here only for processor scheduling. It gives preferential service to high priority tasks among those allocated to the same processor. PR service adds more flexibility to system design, compared with simple FCFS processor queues. Priorities must be carefully chosen. It can also result in adverse effects on performance given the synchronized nature of task communications, e.g., a low priority bottleneck task may seriously reduce task system throughput.

Solutions of SRVN networks containing priority centers are therefore important in the proper representation of such systems. The following sections of the paper present the approximation based upon the Mean Value Analysis, inspired by [BRYA84]. The algorithm for priority service approximations is incorporated into the basic algorithm for performance evaluation of SRVN models with processor contention.

2 List of Symbols

f_i = throughput of task i, messages received/sec.

v_{jp} = mean residual life time of task j in phase p

n_{ijk} = probability that task k is in the queue when task i sends a message to task j

pd_{ip} = mean processor delay (execution plus waiting) by task i, phase p

q_{ip} = utilization of task i, phase p

tp_i = priority of task i (1 is the lowest priority)

y_{ijp} = mean number of RNV calls for task j, generated from task i in phase p

Y_{ij} = total number of RNV calls for task j, generated in all phases of task i, $Y_{ij} = \sum_p y_{ijp}$

s_{ip} = execution time of task i in phase p (total)

ΔY_{ip} = number of processor requests by task i in phase p. These occur at the start of each phase, and following each RNV, so $\Delta Y_{ip} = 1 + \sum_j y_{ijp}$. (In some systems later phases may not begin with a schedule request and ΔY_{ip} should not include the extra unit for $p > 1$.)

$\mu_{ip}^{-1} = s_{ip}/\Delta Y_{ip}$ = execution service time

$\lambda_{ip} = f_i \Delta Y_{ip}$ = rate at which task i joins the processor ready-queue in phase p.

x_{ip} = the time to complete phase p of task i

$w1_{ij}$ = waiting of task i for the end of residual service of task j

$w2_{ij}$ = waiting of task i at task j queue, following $w1_{ij}$

The parameters supplied by the modeller are the task execution times by phase, s_{ip}, the message frequencies y_{ijp}, and the processor allocations and task priorities. The model results are the task throughputs, the total phase times x_{ip}, and the mean wating times both for other tasks ($w1_{ij}, w2_{ij}$) and for the processors (wfp_{ip}).

Because of the synchronization of tasks imposed by the RNV, the task throughputs satisfy the following set of linear equations

$$f_j = \sum_{ip} y_{ijp} f_i, \quad j = R+1, N \tag{1}$$

Tasks that do not receive, but only send messages are (without loss of generality) numbered 1 to R and termed "reference" tasks; they loop indefinitely and drive the system, giving a closed model. By solving (1) we may obtain a set of equations of the following form:

$$f_i = \sum_{r=1}^{R} a_{ir} f_r, \quad i = R+1, \ldots N \tag{2}$$

where the a_{ir} coefficients are constants, found by the solution process.

Solving the model is essentially an iteration of (2) and the following calculations. First,

$$f_r = \frac{1}{\sum_p x_{rp}} \quad r = 1, \ldots R \tag{3}$$

which calculates the loop in frequency of the reference tasks. Following the algorithm given for Basic SRVNs in [WOOD86], we use the following equations in this iteration:

$$x_{ip} = (\Delta Y_{ip}) pd_{ip} + \sum_j y_{ijp} w_{ij} \tag{4}$$

$$w_{ij} = w1_{ij} + w2_{ij} + x_{j1} \tag{5}$$

$$w1_{ij} = \sum_{p=1}^{2} (q_{jp} - f_i Y_{ij} x_{jp} + Prob(E_{ijp}))(v_{jp} + \sum_{u=p+1}^{2} x_{ju}) \tag{6}$$

$$w2_{ij} = \sum_k n_{ijk} \sum_p x_{jp} \tag{7}$$

$$n_{ijk} = f_k Y_{kj}(w_{kj} - x_{j1}) \tag{8}$$

$$v_{jp} = x_{jp} + \sum_k y_{jkp} w^2_{jk} / x_{jp} \tag{9}$$

$$Prob(E_{ij2}) = x_{j2} / (x_{j2} + (f_i Y_{ij})^{-1} - w_{ij}) \tag{10}$$
$$Prob(E_{ij1}) = 0$$

The part in these equations which is new for priorities is the processor delay pd_{ip} in (4), which reflects the effect of priorities and which is determined in the next section. Equation (4) shows the makeup of one phase. Equations (5) - (8) are a mean-waiting-time computation based on iterative mean value analysis, adapted for two or more phases of service. E_{ijp} in Equation (6) is the event of a task having to wait for a phase which it initiated by its own previous message. n_{ijk} is the probability that task k is in the queue for task j; the value given by Equation (8) is further modified in [WOOD86] to account for second-order inter-task interactions.

3 MVA PR Approximation

We analyse the following priority model of the combined multiprocessor/multitask distributed system:

1. The allocations of tasks to processors, and the priorities of tasks, are static. Priority 1 is lowest.
2. Processor scheduling is preemptive-resume.
3. Task message queueing is FCFS.
4. Tasks request RNVs with other tasks randomly and independently, according to predefined probabilities. The average numbers of RNV calls during each phase of each task are known as y_{ijp}.
5. The processor execution of a task is made up of slices, each terminated either with a RNV request or a decision of phase end (final slice of a phase). The time of each slice is exponentially distributed with the same mean for all slices of the same phase, but the mean may be different for different phases.

The approximation (as in [BRYA84]) is based on breaking the queueing delay into two parts:

- a part wfp_{ip} due to tasks in the processor ready queue (or in execution) as it is seen by task i (of priority tp_i), in phase p, arriving for one slice of processor execution,
- a part w^*_{ip} due to arrivals of higher priority tasks and their execution while task i is waiting in the ready queue or is in service.

This gives the mean processor delay per visit by task i phase p as

$$pd_{ip} = \mu_{ip}^{-1} + wfp_{ip} + w_{ip}^{*} \tag{11}$$

Further, it is easily shown that

$$w_{ip}^{*} = pd_{ip}(\sum_{j,u;j\succ i} \lambda_{ju}^{(ip)})\mu_{ju}^{-1} \tag{12}$$

where

$\lambda_{ju}^{(ip)}$ = the conditional mean arrival rate of competing task j in phase u to the ready queue, conditioned on task i phase p being there ahead of it. These are determined below.

$j \succ i$ (read as "j dominates i") denotes a task of higher priority than i, and allocated to the same processor.

Thus we find

$$pd_{ip} = [\mu_{ip}^{-1} + wfp_{ip}]/(1 - \sum_{j,u;j\succ i} \lambda_{ju}^{(ip)}\mu_{ju}^{-1}) \tag{13}$$

The competing throughputs $\lambda_{ju}^{(ip)}$ are found from the steady-state values λ_{ju} and the following considerations. If task j phase u sends messages to task i (i.e. if $y_{jiu} > 0$) then task j may be blocked when i requests the processor. This reduces the task j competing request rate by a fraction $(f_j y_{jiu})/f_i$, provided task i is in phase 1; the component of λ_{ju} that is infeasible has value

$$\begin{aligned} \delta\lambda_{ju}^{(ip)} &= (f_j y_{jiu}\lambda_{ju})/f_i \;\; sec^{-1}, \qquad & p = 1 \\ &= 0 & p > 1 \end{aligned} \tag{14}$$

Similarly if task i sends messages to task j (at rate $f_i Y_{ij}$) and task j is in phase 1, task i is blocked; thus the following task-j request component $\Delta\lambda_{j1}^{(ip)}$ is infeasible at the time of a task-i processor request:

$$\begin{aligned} \Delta\lambda_{ju}^{(ip)} &= f_i Y_{ij}\Delta Y_{j1} \qquad & u = 1 \\ &= 0 & u > 1 \end{aligned} \tag{15}$$

The conditional rates for competing tasks were estimated by subtracting these infeasible components, giving the adjusted values

$$\lambda_{ju} - \Delta_{ju}^{(ip)} - \delta_{ju}^{(ip)} \tag{16}$$

Finally we must include an effect due to processor blocking. When all tasks on a processor are blocked, the processor is idle and reacts only to messages or acknowledgements from outside it. The total rate of these "external events" is defined as Λ_i:

$$\Lambda_i = \sum_{\{j,k|proc(j)=proc(i);proc(k)\neq proc(i)\}} (f_j Y_{jk} + f_k Y_{kj})$$

where $proc(i)$ is the processor which executes task i. Similarly, let Λ_i^{*} be the total rate of processor scheduling events due to phase initiation and to messages between tasks in the same processor; it is given by

$$\Lambda_i^{*} = \sum_{\{j,k|proc(j)=proc(k)=proc(i)\}} f_j Y_{jk} + \sum_j f_j(P_j - 1)$$

Since internal events only occur during periods when the processor is busy, the conditional rate of internal events during these periods id Λ_i^*/U_i, where U_i is the processor utilization (of $proc(i)$). Assuming the external event rate is independent of the processor state, the conditional total rate of scheduler events is $\Lambda_i + \Lambda_i^*/U_i$, which is higher than the average rate.

Clearly the competing rates $\lambda_{ju}^{(ip)}$ are affected by this phenomenon. As an approximation, each rate in (16) was multiplied by the overall average factor

$$K_i = (\Lambda_i + \Lambda_i^*/U_i)/(\Lambda_i + \Lambda_i^*)$$

to account for this effect. Then:

$$\lambda_{ju}^{(ip)} = K_i(\lambda_{ju} - \Delta_{ju}^{(ip)} - \delta_{ju}^{(ip)}) \tag{17}$$

There are two points of approximation in (17); the assumption that all λ_{ju} are affected in the same proportion, and that the external rates are independent of the state of the processor. These approximations are appropriate to the level of detail in our analysis.

Combining (13) and (17) gives, with a little algebra

$$pd_{ip} = (\mu_{ip}^{-1} + wfp_{ip})/(1 - pu_{ip}) \tag{18}$$

$$pu_{ip} = \sum_{j \succ i,u} \lambda_{ju}^{(ip)} \mu_{ju}^{-1} \tag{19}$$

For the computation of wfp_{ip} in (18), a fraction of cases $(1-U_i)\Lambda_i/(\Lambda_i+\Lambda_i^*) = (1-K_iU_i)$ have value zero because they are arrivals during a processor-idle period; we assume each of these arrivals in fact ends such a period. The remaining fraction has a value based on the conditional arrival rates given by (17), and the fact that the probability of finding task j phase u at the processor is $\lambda_{ju}^{(ip)} pd_{ju}$. A standard MVA argument, incorporating the fraction of values which are zero, gives:

$$wfp_{ip} = \sum_{j \succeq i,u} K_i U_i \lambda_{ju}^{(ip)} pd_{ju} \mu_{ju}^{-1} \tag{20}$$

This completes the approximation for pd_{ip}.

An important priority inversion effect happens when two or more high priority tasks request RNVs with a shared set of low priority tasks only and all of them are allocated to the same processor. The fact that the requested tasks have lower priority forces equal throughput of the higher priority tasks, and our approximation does not model this phenomenon. We have ignored it in evaluating the approximation.

4 Examples

The following three examples of SRVN models for distributed software have been chosen to show the results of the approximation. The examples are

- a real-time data base application with three tasks running on two processors (example 1)
- a user-server system executing on two and three processors with strong concentration of traffic to one task server and with processor utilization from very light to over 0.9 (examples 2a and 2b)

- distributed software executing on four processors with more diversified task interactions and two execution phases (example 3).

The evaluation of accuracy of the throughput approximation uses timed Petri-Net and simulation results. First the relative errors E are calculated for each case (set of parameters) of each model. The root mean square error E_{rms} is then calculated for all cases of a particular SRVN model to give an overall estimate of likely errors.

Table 1: Parameters for real-time data base application software

Case	Service time		RNV calls		Priorities	
	s_{12}	s_{22}	y_{132}	y_{232}	T1	T2
1	2	4	2	2	1	1
2	2	4	2	2	1	2
3	2	4	2	2	2	1
4	4	4	1	2	1	1
5	4	4	1	2	1	2
6	4	4	1	2	2	1
7	8	4	1	2	1	1
8	8	4	1	2	1	2
9	8	4	1	2	2	1
10	16	4	1	2	1	1
11	16	4	1	2	1	2
12	16	4	1	2	2	1
13	40	4	1	2	1	1
14	40	4	1	2	1	2
15	40	4	1	2	2	1

The choice of parameters is such that for the same set of task priorities there is a significant variations in the service times. Within each group of three cases we examine equal priority for T1 and T2, priority to the shorter task, and priority to the longer task.

The results of throughput approximation, given in Table 2, are mostly close to the simulation results. The combined error measure E:

$$E = \sum_r |f_r(exact) - f_r(approx.)| / \sum_r f_r(exact)$$

is used to summarize the errors for all the reference tasks in a system. (In this example the sum is over tasks 1 and 2). This compares errors to total throughput, which prevents small errors in small throughputs from swamping the comparison.

The accuracy is quite good except in Cases 9, 12, and 15, where task 2 with progressively heavier computation is given high priority; the approximation depresses the throughput of the low-priority excessively (almost to zero in Case 15). This error is less serious than it might be, since it is usually the short tasks that are given priority rather than the long ones, and when the short tasks have priority in Cases 8, 11 and 14 the accuracy is excellent.

Example 2 Hierarchical User/Server Software

This second example is much more complex because it has server tasks on the same processor with their users. Two versions were considered, a symmetrical version (Example 2a, Tables 3 and 4) and a less symmetrical version (Example 2b. Tables 5 and 6). The tasks have preemptive priority on the processor but FIFO queueing is used for messages to the server tasks, and these two disciplines interact, as was seen by statistical analysis of the

Table 2: Throughput of reference tasks in Example 1.

Case	Task	Simulation	Approximation	Error E [%]
1	1	0.1063 ± 0.0007	0.1147	
	2	0.1020 ± 0.0008	0.1013	4.37
2	1	0.0777 ± 0.0007	0.0741	
	2	0.1208 ± 0.0005	0.1199	2.27
3	1	0.1521 ± 0.0010	0.1489	
	2	0.0740 ± 0.0007	0.0797	3.94
4	1	0.1342 ± 0.0007	0.1296	
	2	0.0835 ± 0.0005	0.0828	2.43
5	1	0.0736 ± 0.0004	0.0655	
	2	0.1408 ± 0.0007	0.1351	6.44
6	1	0.1835 ± 0.0007	0.1851	
	2	0.0364 ± 0.0004	0.0201	8.14
7	1	0.0874 ± 0.0006	0.0853	
	2	0.0576 ± 0.0008	0.0580	1.72
8	1	0.0426 ± 0.0003	0.0397	
	2	0.1418 ± 0.0008	0.1407	2.17
9	1	0.1076 ± 0.0006	0.1092	
	2	0.0199 ± 0.0003	0.0051	12.86
10	1	0.0517 ± 0.0004	0.0507	
	2	0.0358 ± 0.0003	0.0348	2.29
11	1	0.0230 ± 0.0003	0.0222	
	2	0.1453 ± 0.0008	0.1448	0.77
12	1	0.0582 ± 0.0005	0.0586	
	2	0.0103 ± 0.0002	0.0013	13.72
13	1	0.0230 ± 0.0005	0.0229	
	2	0.0166 ± 0.0004	0.0155	3.03
14	1	0.0098 ± 0.0001	0.0095	
	2	0.1474 ± 0.0008	0.1477	0.38
15	1	0.0243 ± 0.0004	0.0244	
	2	0.0042 ± 0.0001	0.0002	14.39
			E_{rms}	7.01%

Table 3: Parameters for symmetrical user/server software

Case	Service time			Priorities				
	s_{41}	s_{51}	s_{61}	T1	T2	T3	T4	T5
1	1	1	1	1	1	1	1	1
2	1	1	1	1	2	3	4	5
3	1	1	1	1	2	3	3	3
4	0.5	0.5	1	1	1	1	1	1
5	0.5	0.5	1	1	2	3	4	5
6	0.5	0.5	1	1	2	3	3	3
7	0.5	0.5	0.5	1	1	1	1	1
8	0.5	0.5	0.5	1	2	3	4	5
9	0.5	0.5	0.5	1	2	3	3	3

queues in the simulation. However the approximation for each queue ignores the disciplines at these queues, and probably this is the source of rather large errors in Table 4, between 20 and 30%.

Table 4: Throughput - comparison of simulation and approximation results for symmetrical user/server software

Case	Task	Simulation	Approximation	Error [%]
1	1	0.0806 ± 0.0004	0.0751	
	2	0.0807 ± 0.0004	0.0751	
	3	0.0805 ± 0.0005	0.0751	13.98
2	1	0.0508 ± 0.0004	0.0636	
	2	0.0717 ± 0.0002	0.0753	
	3	0.1351 ± 0.0004	0.0838	26.28
3	1	0.0504 ± 0.0006	0.0623	
	2	0.0700 ± 0.0004	0.0737	
	3	0.1322 ± 0.0003	0.0897	23.00
4	1	0.1014 ± 0.0007	0.0936	
	2	0.1019 ± 0.0007	0.0936	
	3	0.1019 ± 0.0012	0.0936	7.99
5	1	0.0680 ± 0.0002	0.0825	
	2	0.0930 ± 0.0003	0.0983	
	3	0.1683 ± 0.0004	0.1088	24.08
6	1	0.0681 ± 0.0009	0.0811	
	2	0.0910 ± 0.0011	0.0958	
	3	0.1634 ± 0.0013	0.1109	21.80
7	1	0.1340 ± 0.0010	0.1256	
	2	0.1330 ± 0.0010	0.1256	
	3	0.1340 ± 0.0010	0.1256	6.03
8	1	0.0643 ± 0.0010	0.0831	
	2	0.1097 ± 0.0013	0.1374	
	3	0.2619 ± 0.0019	0.1799	29.48
9	1	0.0649 ± 0.0007	0.0826	
	2	0.1083 ± 0.0012	0.1305	
	3	0.2547 ± 0.0011	0.1815	26.43
			E_{rms}	21.05 %

Throughputs tend to be underestimated in both this and the previous example, apparently because of intertask arrival dependencies induced by the sequencing of at the server tasks. However the correct general relationship between low-priority and high-priority throughputs is maintained.

Experiments with other sets of parameters indicate that the errors in Table 4 are worse than usual. Just using a more asymmetrical set of execution parameters, as in Example 2b, shows a substantial improvement in errors so that most are under 20%. Tables 5 and 6 give the parameters and results.

Example 3. Hierarchical Distributed Software

The software composed of seven tasks is executed on four processors. The send/receive graph in Figure 5 shows the hierarchical structure as in previous examples, but there is not so strong contention for a server as before. Effectively there are three server tasks 3, 6 and 7, executing each on a different processor. All workload related parameters are constant as shown in Fig.4. The only variables are priorities, given in Table 7.

Table 5: Parameters for asymmetrical user/server software

Case	Service time s_{61}	Priorities				
		Processor 1			Processor 2	
1	10.	1	1	1	1	1
2	1.	1	1	1	1	1
3	0.1	1	1	1	1	1
4	10.	2	1	3	1	1
5	1.	2	1	3	1	1
6	0.1	2	1	3	1	1
7	10.	2	1	3	1	2
8	1.	2	1	3	1	2
9	0.1	2	1	3	1	2
10	10.	2	3	1	1	1
11	1.	2	3	1	1	1
12	0.1	2	3	1	1	1

The approximation errors shown in Table 8 are in a similar range of 10–20%, again probably because of interdependencies induced by the messaging. As an example, tasks 3 and 4 are called by task 1, but only one at a time. Thus the effective number of tasks that are competing for processor 2 execution is always 2, not 3 as taken into the calculations.

5 Conclusions

The proposed Mean-Value Analysis approximation for performance of software systems with preemption and processor contention, in general gives only moderately good accuracy of results. The overall error ranges from a few percent up to nearly 30% in some cases. Other examples showed errors similar to the four shown here, and mostly under 20%. This accuracy may be useful for preliminary analysis of designs.

This is an extremely challenging problem for MVA, with a very rich set of possible dependencies between tasks. The approximations described here are a compromise between accuracy and detail, based purely on heuristic considerations and extensive experimentation.

The works accuracy was obtained for low priority tasks on a heavily loaded processor; the approximation was very pessimistic and under-estimated their throughput drastically. Care should be taken over using the approximation in these cases, or in cases with priority inversion (where high priority tasks send messages to low priority tasks, and most wait for them).

Further work is underway to analyze logical task interactions first, before applying expressions based on stochastic task behaviour.

6 References

[BRYA84] R.M. Bryant, A.E. Krzesinski, M.S. Lakhsmi, K.M. Chandy, "The MVA Priority Approximation", *ACM Transactions on Computer Systems*, vol. 2, no. 4, November 1984, pp. 335–359.

[CHER83] D.R. Cheriton, "The V Distributed System", Communications of the ACM, vol. 31, No. 3, March 1988.

Table 6: Throughput – comparison of simulation and approximation results for asymmetric user/server software

Case	Task	Simulation	Approximation	Error[%]
1	1	0.0104 $\pm$ 0.0001	0.00822	
	2	0.0075 $\pm$ 0.0002	0.00619	
	3	0.0152 $\pm$ 0.0003	0.0125	18.95
2	1	0.0498 $\pm$ 0.0005	0.0476	
	2	0.0365 $\pm$ 0.0006	0.0344	
	3	0.0737 $\pm$ 0.0009	0.0713	4.19
3	1	0.0661 $\pm$ 0.0005	0.0601	
	2	0.0488 $\pm$ 0.0004	0.0434	
	3	0.0990 $\pm$ 0.0008	0.0905	9.30
4	1	0.0104 $\pm$ 0.0002	0.00822	
	2	0.0071 $\pm$ 0.0002	0.00617	
	3	0.0160 $\pm$ 0.0003	0.0125	19.73
5	1	0.0502 $\pm$ 0.0005	0.0478	
	2	0.0335 $\pm$ 0.0003	0.0338	
	3	0.0788 $\pm$ 0.0012	0.0721	5.78
6	1	0.0671 $\pm$ 0.0003	0.0604	
	2	0.0441 $\pm$ 0.0003	0.0423	
	3	0.1094 $\pm$ 0.0006	0.0921	11.70
7	1	0.0107 $\pm$ 0.0002	0.00822	
	2	0.0070 $\pm$ 0.0001	0.00617	
	3	0.0159 $\pm$ 0.0004	0.0126	19.62
8	1	0.0573 $\pm$ 0.0009	0.0504	
	2	0.0333 $\pm$ 0.0007	0.0319	
	3	0.0781 $\pm$ 0.0007	0.0678	11.03
9	1	0.0824 $\pm$ 0.0004	0.0680	
	2	0.0419 $\pm$ 0.0004	0.0384	
	3	0.1038 $\pm$ 0.0007	0.0831	16.92
10	1	0.0103 $\pm$ 0.0001	0.00821	
	2	0.0080 $\pm$ 0.0002	0.0062	
	3	0.0145 $\pm$ 0.0003	0.0124	18.26
11	1	0.0498 $\pm$ 0.0006	0.0474	
	2	0.0383 $\pm$ 0.0004	0.0348	
	3	0.0695 $\pm$ 0.0007	0.0702	4.19
12	1	0.0655 $\pm$ 0.0004	0.0598	
	2	0.0522 $\pm$ 0.0003	0.0441	
	3	0.0929 $\pm$ 0.0007	0.0887	8.55
				E_{rms} = 13.66 %

Table 7: Priorities of tasks in Example 3.

Case	Priorities				
	Processor 1		Processor 2		
1	1	1	1	1	1
2	2	1	1	1	2
3	2	1	2	2	1
4	1	2	2	2	1
5	1	2	1	1	2

Table 8: Throughput - comparison of simulation and approximation results for hierarchical distributed software

Case	Task	Simulation	Approximation	Error E [%]
1	1	0.1362 ± 0.0028	0.1224	
	2	0.0616 ± 0.0024	0.0583	8.65
2	1	0.1498 ± 0.0027	0.1151	
	2	0.0676 ± 0.0016	0.0678	16.05
3	1	0.1703 ± 0.0023	0.1452	
	2	0.0444 ± 0.0015	0.0373	15.00
4	1	0.1357 ± 0.0014	0.1207	
	2	0.0567 ± 0.0011	0.0469	12.89
5	1	0.1166 ± 0.0015	0.0860	
	2	0.0792 ± 0.0015	0.0747	17.93
				$E_{rms} = 14.46$ %

[DAIG81] J.M. Daigle and C.E. Houstis, "Analysis of a Task Oriented Multipriority Queueing System", IEEE Transactions on Communications, vol. COM-29, No. 11, November 1981.

[DOD83] U.S. Dept. of Defence, "Reference Manual for the Ada Programming Language", MIL-STD-1815a, 1983.

[GAMM84] N. Gammage, R. Kamel, J. Perry, "XMS System Concepts and Architecture", *Telesis*, 1984 three, pp 20-25.

[HEID83] P. Heidelberger, K.S. Trivedi, "Analytic Queueing Models for Programs with Internal Concurrency", IEEE Trans. on Comp., vol18z. C-32, No. 1, pp. 73-82. January 1983.

[HOLL87] M.A. Holliday, M.K. Vernon, "A Generalized Timed Petri Net Model for Performance Analysis", IEEE Trans. on Software Engineering, v.SE-13, No. 12, pp. 1297-1310, December 1987.

[KANT87] K. Kant, "Modelling Interprocess Communication in Distributed Programs", Proc. Int. Workshop on Petri Nets and Performance Models, Madison, Wisc., August 1987,

[MARS84] M. Ajmone Marsan, G. Balbo, and G. Conte, "A Class of Generalized Stochastic Petri Nets for the performance Evaluation of Multiprocessor Systems", ACM Trans. on Computer Systems, vol. 2, No. 2, May 1984.

[MIER88] J.W. Miernik, C.M. Woodside, J.E. Neilson, D.C. Petriu, "Throughput of Stochastic Rendezvous Networks with Caller-Specific Service and Processor Contention", *Proceedings, IEEE INFOCOM '88*, pp 1040-1049, New Orleans, USA, March 1988.

[RADI84] S. Radia, "A Markov Model for Distributed Communicating Sequential Processes", Performance 84, (ed. E. Gelenbe), pp. 3-16, Elsevier, 1984.

[SHAT88] S.M. Shatz and W.K. Cheng, "A Petri Net Framework for Automated Static Analysis of Ada Tasking Behavior", to appear in the Journal of Systems and Software 1988. pp. 75-83.

[WOOD86] C.M. Woodside, "Throughput Calculation for Basic Stochastic Rendezvous Networks", Report SCE-86-9, Second Revision, November 1987. To appear in Performance Evaluation.