Performance of Concurrent Rendezvous Systems with Complex Pipeline Structures

Real-Time and Distributed Systems Group
Carleton University, Ottawa, Canada
cmw@sce.carleton.ca

February 11, 1998

Abstract

The term "complex pipeline" describes a set of tasks which process incoming data in a sequence, like a pipeline, but have various kinds of parallel execution steps coupled into the main stream of execution. Examples are, splitting off of parallel streams, and shared server tasks. Examples are found in processing to interpret radar data, and other real-time systems. Rendezvous systems like Ada have static tasks, static processor allocations and synchronous inter-task communications, which can cause potential performance problems. The growing importance of rendezvous-based environments, including Ada, requires that we be able to predict these problems. Models such as Petri nets, are often too expensive to solve; fast approximation techniques are needed. The approach of "stochastic rendezvous networks" is adapted here to deal with complex pipelines. This paper describes an algorithm and evaluates its accuracy; the algorithm is the major feature of the paper, including a "Conditional Mean Value Analysis" step. It includes processor queueing, which was not modelled in the earlier work. The method is several orders of magnitude faster than Petri net analysis even on small examples. The accuracy obtained is generally better than 10%.

1 Introduction

In concurrent software written for multi-tasking, multi-processing environments such as Ada [DOD83], V [CHER83], or XMS [GAMM84], we often find a sequence of tasks operating as a pipeline on a stream of data. For example, a radar tracking system might have tasks for filtering, detection, track correlation, track update, display and storage updating for the track. It may be a simple sequence of tasks as illustrated in Figure 1(a) or a more complex pattern which we will term a "complex task pipeline".

Pipelines are introduced in hardware to obtain extra throughput by overlapping successive operations on successive items of data, and task pipelines can have the same effect when they are executed on a succesion of processors. Performance calculations and optimizations for deterministic clocked hardware pipelines are surveyed by Kogge [KOGG81].

However the previous studies miss out some important features of task pipelines. First, intertask communications may affect performance, particularly in a rendezvous system where there is no intertask buffering of communications. Second, it is common for task architectures to have the more complex structures already mentioned, perhaps with branching paths or shared resources. Examples of resources shared between stages are, a shared file, or a buffer pool, and in a rendezvous system they are likely to be managed by a separate task called a *server task*. Pipeline tasks must queue for access to server tasks, and in an extreme case there may be several levels of servers involved.

This paper shall define a complex task pipeline to be:

- a sequence or tree of tasks, possibly with branches in the flow to create parallel segments,

- data dependent (ie. stochastic) access to server tasks providing shared services or managing shared resources.

The analysis here will cover the features in the first item, leaving server tasks to a later paper. The complex pipelines will be made up of software tasks statically allocated to processors and communicating via rendezvous, interpreted as a send-receive-process-reply sequence (see Section 2). There may be waiting for a rendezvous or for processor scheduling; this waiting is estimated in order to determine the pipeline throughput capacity. Response times are also obtained as a by-product of the analysis.

This paper extends the method introduced in [WOOD84] and [WOOD88], by explicitly modelling processor contention sets and deterministic communication sequences along the pipeline. It also introduces a new method for analysis of queues in systems with arbitrary synchronization and static dependencies, called "Conditional Mean Value Analysis". The essence of the contribution of this paper is fast approximations for contention. With this method, systems of over 80 tasks have been modelled.

For tasks with rendezvous, timed Petri Nets have been used in [WOOD88], [SCHA88], and [KANT88]. In [WOOD88] the Generalized Stochastic Petri Net (GSPN) model of Ajmone Marsan, Balbo et al [AJMO84] was applied to a class of stochastic models of task behaviour called Stochastic Rendezvous Nets. Kant [KANT88] used a combination of timed Petri Nets and queues for a similar class of models. Other authors have used Petri nets to represent and analyze the flow of control between tasks in Ada programs. Schatz [SCHA88] proposed doing this, based directly on parsing of Ada code, followed by timed Petri Net modelling of performance. However he does not include modelling of processor dispatching delays.

In any case, it seems (see for instance [WOOD88] for a discussion) that Petri Net models are too large and computation-intensive for use on systems of practical scale, although hierarchical decomposition may overcome this obstacle. The present approach combines a degree of hierarchical decomposition with a delay approximation. It is a special case of approximation of Timed Petri Nets or other equivalent models by higher level constructs, with approximate delays for the subsystems. This is the purpose also of the Active Server Network family of algorithms described in [WOOD84] and applied to Stochastic Rendezvous Networks in [WOOD88]. The approach uses ideas from queueing network analysis, notably surrogate delays [JACO82] and approximate Mean Value Analysis [REIS79].

The Rendezvous Network Approach

The Rendezvous (abbreviated RNV hereafter) Network modelling approach identifies all the various waiting delays and approximates them by calculations adapted from approximate Mean Value Analysis [REIS79]. In the pipeline, each task waits for the following task to complete the RNV which passes on the data, plus there may be processor waiting and waiting for shared server tasks. In adapting the Mean Value calculation, conditions governing the state of the accepting task or processor are considered. The method reflects saturation of the pipeline back towards the beginning, and throughput is determined by how fast the source task can cycle.

The present research extends the previous work on stochastic sequences of rendezvous by examining deterministic sequences which occur along the pipeline, as well as processor contention. Shared server tasks are then a natural generalization which will be obtained by combining this and the earlier work. The goal of this paper is to show that the approach can be extended, and to evaluate its accuracy. The principal advantage of the method is speed which is several orders of magnitude faster than Petri net analysis even on small examples. The accuracy obtained is only modest, with errors generally under 10% but occasionally up to twice that.

Section 2 begins with a special model for simple pipelines, and Section 3 describes contention for processors by the pipeline tasks. Section 4 adds splits in the pipeline flow, to give the complete model. The approximations are compared to exact results found from GSPN analysis of the smaller examples and to simulation results for the others.

2 Simple Task Pipelines With Rendezvous

To introduce the approach to be taken for complex pipelines, first consider the more usual "simple task pipeline", illustrated in Figure 1(a) and described as follows:

- a sequence of N static tasks communicating in one direction by rendezvous, in pipeline style. The first task is a source, computing and generating data for the remainder of the pipeline, while the last task is a sink (after completing its computation).
- each task has a processor to itself (this will be relaxed shortly to allow tasks to be co-allocated), and there are no contention delays in communications. Communication delays can be included in the task execution times.
- each task executes in a cycle divided into three phases:

  Phase 1 (synchronous): (a) receive input data as a message, (b) check the data, (c) reply to sender, to acknowledge completion of the handshake;

  Phase 2 (asynchronous execution): execute the task function on the data, for this stage of the pipeline.

  Phase 3 (transmit by RNV) (a) send the data to next task in the pipeline, (b) wait for the reply, (c) make ready to receive the next input message (cycle back to phase 1)
- task 1, the source, omits phase 1 and task N, the sink, omits phase 3 in its cycle.

This pipeline always has input waiting and runs at its maximum rate. Poisson arrivals at a lower rate are a simple adaptation, described at the end. The following notation describes the sequence of activities in a single RNV and is illustrated in Figure 1(b):

f =throughput or frequency of task cycling for all tasks in the pipeline; s_{ip} =mean execution time of task i in phase p; x_{ip} =mean duration of phase p of task i. For $p = 1, 2$, a phase consists purely of execution, and $x_{ip} = s_{ip}$. w_{ij} =mean delay to task i in a RNV with task j. y_{ijp} =the mean number of of messages sent to task j by task i in phase p. In this simplest version $y_{ij1} = y_{ij2} = 0$. y_{ij3} =1 if task j succeeds task i in the pipeline, or 0 otherwise (i.e., $y_{i,i+1,3} = 1$).

The notation $\tilde{s}_{ip}$, $\tilde{x}_{ip}$, $\tilde{w}_{ij}$, etc. will be used for the random variable whose mean is s_{ip}, x_{ip}, w_{ij}.

Now we shall determine the delays experienced by a task in processing and in waiting at the RNVs. The only way a task can be made to wait is, if the successor task is still busy from the previous message. That is, the only reason task j can make task $j-1$ wait is if task j is busy in its second or third phase. Define:

E_{jp} =event that, at the moment task $j-1$ sends to task j, the latter is busy in phase p, $p > 2$. E_j =the union of E_{j2} and E_{j3}; task $j-1$ 'overtakes' task j. V_{jp} =the mean residual life of phase p of task j, called $MRL(\tilde{x}_{jp})$. In general, $MRL(\tilde{x}) = Exp\{\tilde{x}^2\}/2Exp\{\tilde{x}\}$ (see [KLEI75] Sec. 5.2 for a discussion of mean residual life). If $\tilde{x}$ is exponentially distributed, $MRL(\tilde{x}) = Exp\{\tilde{x}\}$.

Then for a simple pipeline the delays defined above are given by:

$$w_{j-1,j} = x_{j1} + Prob\{E_{j2}\}(V_{j2} + x_{j3}) + Prob\{E_{j3}\}V_{j3}, \quad j = 2, \ldots, N \tag{1}$$

$$x_{j1} = s_{j1}, \quad x_{j2} = s_{j2}, \quad x_{j3} = w_{j,j+1}, \quad j = 1, \ldots, N \tag{2}$$

with the exceptions that $x_{11} = x_{N3} = 0$. The MRL values are

$$V_{jp} = Exp\{(\tilde{x}_{jp})^2\}/2(x_{jp}), \quad j = 2, \ldots, N \tag{3}$$

Two approximations will be used in (1), one for V_jp and one for Prob$\{E_{jp}\}$. The mean residual life value V is computed on the basis of an assumption that the request instant is random and independent of the state of task j. It is also assumed that $\tilde{w}$ is exponentially distributed, which means that (3) gives:

$$V_{j2} = Exp\{\tilde{s}_{j2}^2\}/2s_{j2}; \quad V_{j3} = w_{j,j+1} \quad j = 1, \ldots, N \tag{4}$$

For $j = N$, we use $V_{N3} = w_{N,N+1} = 0$. MRL values are not needed for phase 1.

The event E_{j2}, that task j is found to be busy in phase 2, is illustrated in Figure 2(a). The probability was found by analyzing the race between task j finishing this phase, and the generation of the next RNV request by task $j-1$. Let $\tilde{t}_{j-1}$ denote the time from the end of the previous RNV to the instant of the next request from task $j-1$; it has mean value $1/f - w_{j-1,j}$. Then:

$$Prob\{E_{jp}\} = Prob\{\tilde{t}_{j-1} < (\tilde{x}_{j2} + \ldots + \tilde{x}_{jp})\}; \quad p = 2 \text{ or } 3$$

which depends on the distributions of $\tilde{t}_{j-1}$ and of $(\tilde{x}_{j2} + \ldots + \tilde{x}_{jp})$. Figure 2(b) shows the random variables involved. An especially simple result is obtained if the distributions are exponential, so for the sake of calculation this was assumed. With a little manipulation this gives the approximation:

$$Prob\{E_{j2}\} = (x_{j2})/(x_{j2} + (1/f) - w_{j-1,j}) \quad (5a)$$

$$\begin{aligned} Prob\{E_{j3}\} &= (1 - Prob\{E_{j2}\})Prob\{E_{j3}|\bar{E}_{j2}\} \\ &= (1 - Prob\{E_{j2}\})x_{j3}/(x_{j3} + (1/f) - w_{j-1,j}) \quad (5b) \end{aligned}$$

$$Prob\{E_j\} = Prob\{E_{j2}\} + Prob\{E_{j3}\} \quad (5c)$$

Now (1)-(3) can be solved for the delay for each task in terms of the execution of the task which follows it. In order to solve for the throughput, the calculations for x and w are applied iteratively. At each iteration the overall throughput f is calculated from the time it takes the source task to cycle:

$$f = 1/(x_{12} + x_{13}) \tag{6}$$

This throughput value is then used to find the other quantities for the next iteration, using the following algorithm. The starting value for f was taken as a bounding value found from a task saturation condition.

Algorithm 1, for Simple Pipelines

Step 1 – initialize $f = 1/max_i(s_{i1} + s_{i2})$; $x_{i1} = s_{i1}$, $x_{i2} = s_{i2}$, for $i = 1, .., N$; and V_N from (4).

Step 2 – for each value of j in the order $j = N-1, N-2, \ldots, 1$ determine $w_{j,j+1}$, x_{j3}, V_{j2} and V_{j3} using (1), (2), (4), (5).

Step 3 – set $f = 1/(x_{12} + x_{13})$. If within a preset percentage of the previous value, stop, else repeat from 2.

Examples and Accuracy

The algorithm was applied to the example of four tasks shown in Figure 1(a), and compared to an exact throughput analysis found by a Markov Chain analysis of a GSPN model [AJMO84]. For this purpose the distributions of execution times were taken to be exponential, although the algorithm is not limited to such cases. The parameter values and results are shown in the following table.

Table 1: Simple Pipeline Throughputs (No Processor Contention)

Four tasks as in Figure 1(a) with default parameters: $s_{i1} = 0.1$sec, $s_{i2} = 1.0$ sec, $y_{i,i+1,3} = 1$, all i.

Case	Parameters (differences from defaults)	Exact Throughput	Approximate Throughput
1	Default	0.474	0.461
2	$s_{12} = \frac{2}{3}$, $s_{22} = 2$	0.368	0.361
3	$s_{12} = \frac{1}{2}$, $s_{22} = \frac{2}{3}$ $s_{32} = 2.$	0.384	0.376
4	$s_{12} = 0.5$, $s_{22} = 0.8$ $s_{32} = 1.2$ $s_{42} = 1.5$	0.432	0.423
5	$s_{12} = 1.5$, $s_{22} = 1.2$ $s_{32} = 0.8$ $s_{42} = 0.5$	0.432	0.424
6	$s_{12} = 0.5$, $s_{22} = 1.5$, $s_{32} = 1.5$, $s_{42} = 0.5$	0.400	0.391

Case 1 is balanced, while cases 2 and 3 show various unbalanced patterns. Cases 4,5,6 have the same total workload as case 1 but redistributed systematically in increasing order (case 4), in decreasing order (case 5), and increasing first, then decreasing (case 6). The most balanced case, case 1, gives the largest error of about -2.7%, and all the approximate throughputs are less than the exact values. The nominal saturation rate can be found from the longest processing time s_{i2}, plus 0.1 sec. for communication in phase 1 (input) and another 0.1 in the RNV on output. This gives $1.2^{-1} = 0.833$ in case 1, $2.2^{-2} = 0.454$ in cases 2 and 3, and $1.7^{-1} = 0.588$ in cases 4, 5 and 6. The actual value is less than this because of random variations and the associated waiting caused by them and propagated by the rendezvous.

Notice that each task is busy only about half the time (execution time of 1.1 sec per cycle times rate of about 0.4 cycles per sec.). The rest of the time it is blocked, either waiting for input or waiting to pass on its output. This is a penalty of the rendezvous communications, which can be mitigated by introducing buffers between stages.

3 Processor Contention

The allocation of tasks to processors and the resulting queueing delays from the time a task is ready until it begins execution, are now considered. The notation $J(i)$ is used for the processor which executes task i. We suppose that when a task is ready to run it "makes a request" to its processor, is queued and served and "returns" at the end of processing. Processor contention only occurs when a task blocks and then resumes. This always occurs in phase 3 for passing on data, except in task N. It also occurs sometimes in phase 1 of tasks $2 - N$, when even the message from the previous task is not available. This condition "message from task $i-1$ is not available when task i tries to receive it" has already been identified as the event E_i. The mean waiting for a single request depends on the loading of processor $J(i)$, described as the set K_i of tasks contending for the processor. We shall consider a certain degree of state dependence in K_i, for if task $i-1$ is allocated to the same processor and if it is suspended attempting to RNV with task i, then it is removed from contention. Because these processor requests occur at the end of phase 3 of task i, this situation corresponds to the event E_i defined in the last Section, when task $i-1$ is ready before task i is ready. Thus we denote the contention set as K_{iE}:

$$K_{iE} = \{k|J(k) = J(i); k \neq i; k \neq i-1\} \qquad (7)$$

and the corresponding mean wait for the processor is denoted W_{iE}. The opposite condition, in which task $i-1$ is still running and in contention, has contention set $K_{i\bar{E}}$ and mean wait $W_{i\bar{E}}$, with:

$$K_{i\bar{E}} = \{k|J(k) = J(i); k \neq i\} \qquad (8)$$

At a randomly chosen processor request instant the mean wait for the processor is the combination;

$$Processor\_waiting = Prob\{E_i\}W_{iE} + Prob\{\bar{E}_i\}W_{i\bar{E}} \qquad (9)$$

Phase 3 always contains a processor request, for $i \leq N-1$, so:

$$x_{i3} = w_{i,i+1} + Prob\{\bar{E}_i\}W_{i\bar{E}} + Prob\{E_i\}W_{iE}; \quad i = 1, \ldots, N-1 \qquad (10)$$

Phase 1 contains a processor request only in the event $\bar{E}_i$, for $i \geq 2$, and phase 2 never does, so

$$x_{i1} = s_{i1} + \mathrm{Prob}\{\bar{E}_i\}W_{iE}; \quad i = 2, \ldots, N \qquad (11)$$

$$x_{i2} = s_{i2}; \quad i = 1, \ldots, N$$

and

$$x_{11} = x_{N3} = 0$$

as before.

Conditional Mean Value Analysis for Processor Delay

The equation (9) with the condition E_i will now be used in a "Conditional Mean Value Analysis" calculation for *Processor-waiting.* The general form for a "Conditional MVA" model with conditions $C_1, \ldots, C_n$ (which depend in turn on various system state probabilities denoted $state - probs$) is:

$$waiting\ values = \sum_i \{waiting \mid C_i\} Prob\{C_i\} \quad (A)$$

$$state - probs \leftarrow waiting\ values \quad (B)$$

$$Prob\{C_i\} \leftarrow state - probs \quad (C)$$

and the solution approach used here is iterative.

The following development specifically for pipelines uses Eq. (5) for (B) and (C) together, and uses Eq. (9) together with the analysis to follow in the role of (A). Considering W_{iE} first, it is broken into two components, for the task-in-service and the queue:

$$W_{iE} = Pwait_{iE}(1) + Pwait_{iE}(2) \tag{12}$$

where

$Pwait_{iE}(1)$ = waiting due to the task in execution at processor $J(i)$ at the time task i is ready to run,

$Pwait_{iE}(2)$ = waiting due to tasks which are in the queue for processor $J(i)$ at the time task i is ready to run.

We will continue to assume a randomly chosen instant for making the request for execution, subject to some qualifications to be made below. Define:

$E_{iE,k}$ = the event that a request by task i finds task k being executed by processor $J(i)$, given the contention set K_{iE}. Its probability is

$$\text{Prob}\{E_{iE,k}\} = f(s_{k1} + s_{k2}) \text{ for all } k \in K_{iE} \text{ and zero otherwise}$$

$F_{iE,k}$ = the event that a request by task i finds task k waiting in the processor queue, given contention set K_{iE};

σ_k = mean residual life of the processor execution time of task k, $MRL(\tilde{s}_{k1} + \tilde{s}_{k2})$. From the definition of MRL,

$$\sigma_k = (Exp\{\tilde{s}_{k1}^2\} + Exp\{\tilde{s}_{k2}^2\} + 2s_{k1}s_{k2})/2(s_{k1} + s_{k2}) \tag{13}$$

Then the first component of processor waiting is:

$$Pwait_{iE}(1) = \sum_{k \in K_{iE}} \text{Prob}\{E_{iE,k}\}\sigma_k = f \sum_{k \in K_{iE}} (s_{k1} + s_{k2})\sigma_k \tag{14}$$

The second component $Pwait_{iE}(2)$ is the delay for tasks in the processor queue at the time of the request, and is given by:

$$Pwait_{iE}(2) = \sum_{k \in K_{iE}} \text{Prob}\{F_{iE,k}\}(s_{k1} + s_{k2}) \tag{15}$$

To find Prob $\{F_{iE,k}\}$, consider the queue for processor $J(i)$. At a randomly selected instant the probability that task k is in the queue (waiting, not executing) is given by Little's result and by (10) and (11) as

$$\text{Prob}\{\textit{task k in the queue}\} = f \times (\textit{mean processor waiting by task k per cycle})$$

$$= f(W_{kE} + \text{Prob}\{\bar{E}_k\}W_{k\bar{E}}) \tag{16}$$

However this must be adjusted to account for the following subtle but important interaction: task k only waits when some other task is executing and on some occasions it is task i which it waits for. In the calculation of $Pwait_{iE}(2)$ we must exclude these occasions, since task i cannot be simultaneously both in execution and arriving to the processor queue. To adjust, we condition on the task in execution being some other task and not task i. The conditional probability is taken to be a ratio of partial processor utilizations by the tasks, and gives:

$$Prob\{F_{iE,k}\} = \text{Prob}\{\textit{task } k \textit{ in the queue}\}\text{Prob}\{\textit{not task } i \textit{ in execution} \mid \textit{task } k\}$$

$$= f(W_{kE} + \text{Prob}\{\bar{E}_k\}W_{k\bar{E}})\frac{\sum_{l\in K_{iE}, l\neq k} u_l}{u_i + \sum_{l\in K_{iE}, l\neq k} u_l} \qquad (17)$$

where $u_i = f(s_{i1} + s_{i2})$ is the partial utilization of processor $J(i)$ by task i. This adjustment is important because, if there are only two tasks allocated to processor $J(i)$, the right side is zero and $Pwait(2)$ is zero. When one task makes a request the other must be executing, or blocked and not ready; it cannot be queued.

This completes the determination of W_{iE} using (12) with (14) - (17). $W_{i\bar{E}}$ is found by the same process with $\bar{E}$ substituted for E in (12), (14), (15), and using (16) unchanged. Then Eq. (17) is replaced for case $\bar{E}$ by:

$$Prob\{F_{i\bar{E},k}\} = f(W_{kE} + \text{Prob}\{\bar{E}_k\}W_{k\bar{E}})\frac{\sum_{l\in K_{i\bar{E}}, l\neq k} u_l}{u_i + \sum_{l\in K_{i\bar{E}}, l\neq k} u_l} \qquad (18)$$

Processor delay terms also enter the computation of the mean residual life V_{jp}. We will use the Appendix with $z_1 = w_{j,j+1}$ and $z_2 = W_{j,E}$ (since every waiting of task $j-1$ for task j involves event E_j) and the processor waiting will be assumed to be exponentially distributed. Then Eq. (4) is replaced with:

$$V_{j2} = MRL\{\tilde{s}_{j2}\}$$

$$V_{j3} = MRL(\tilde{w}_{j,j+1} + \tilde{w}_{j,E}) = \frac{w_{j,j+1}^2 + W_{jE}^2 + (w_{j,j+1} + W_{jE})^2}{2(w_{j,j+1} + W_{jE})},$$

$$j = 1, \ldots, N-1 \qquad (19)$$

The computation of the throughput is similar to Section 2, with Eqs. (10), (11) in place of (2), and W and V computed as above. Note that, if a task has a processor to itself the W terms are zero and the analysis of Section 2 can be used for that task, ignoring the existence of the processor.

The algorithm for throughputs is similar to the algorithm in section 2, but with the processor delay calculations included in Step 2.

Examples and Accuracy

Examples were computed for the same four-task pipeline and for three different processor allocations, as shown in Figure 3(a),(b),(c). Cases 1–6 have the same task parameters as Cases 1–6 in Table 1, and these are repeated for the two other allocations, giving 18 cases in all. Table 1 can be considered a fourth allocation with one processor per task.

Cases 1, 4, 5 and 6 all have the same total execution time, equally balanced between the two processors but distributed differently among the tasks. The highest throughput is found in case 5, where the tasks are ordered by execution time s_{12}, longest task first; the worst is case 4, ordered the opposite way, and the balanced cases (cases 1 and 6) are between them. The approximation has the most error with the balanced case, as before.

When only one processor is used, all the corresponding cases (i.e. 7, 10, 11, 12) have the same throughput; the processor is the only constraint. With three processors (in cases 13-18) the middle one is more heavily loaded than the others and is the dominant influence on throughput.

Introducing the processor contention delays has increased the errors somewhat; they range from -8.5% in case 15 to +9.1% in case 3. The RMS percentage error is 3.6% and the mean is -0.6%.

Table 2: Simple Pipeline Throughputs with Processor Contention
Four tasks as in Figure 1(a) with default parameters: $s_{i1} = 0.1$, $s_{i2} = 1.0$, $y_{i,i+1,3} = 1$, all $i = 1,..4$. The processor allocation is given by a list $J(1)$, $J(2)$, $J(3)$, $J(4)$.

Case	Parameters (differences from defaults)	Processor Allocation	Exact Throughput	Approximate Throughput
1	Default	1,1,2,2	0.367	0.369
2	$s_{12} = \frac{2}{3}$, $s_{22} = 2$	1,1,2,2	0.302	0.282
3	$s_{12} = \frac{1}{2}$, $s_{22} = \frac{2}{3}$ $s_{32} = 2.$	1,1,2,2	0.304	0.332
4	$s_{12} = 0.5$, $s_{22} = 0.8$ $s_{32} = 1.2$ $s_{42} = 1.5$	1,1,2,2	0.328	0.331
5	$s_{12} = 1.5$, $s_{22} = 1.2$ $s_{32} = 0.8$ $s_{42} = 0.5$	1,1,2,2	0.389	0.339
6	$s_{12} = 0.5$, $s_{22} = 1.5$ $s_{32} = 1.5$ $s_{42} = 0.5$	1,1,2,2	0.347	0.343
7	Default	1,1,1,1	0.233	0.239
8	$s_{12} = \frac{2}{3}$, $s_{22} = 2$	1,1,1,1	0.201	0.200
9	$s_{12} = 0.5$, $s_{22} = 0.6667$ $s_{32} = 2.$	1,1,1,1	0.224	0.219
10	$s_{12} = 0.5$, $s_{22} = 0.8$ $s_{32} = 1.2$ $s_{42} = 1.5$	1,1,1,1	0.233	0.229
11	$s_{12} = 1.5$, $s_{22} = 1.2$ $s_{32} = 0.8$ $s_{42} = 0.5$	1,1,1,1	0.233	0.232
12	$s_{12} = 0.5$, $s_{22} = 1.5$ $s_{32} = 1.5$ $s_{42} = 0.5$	1,1,1,1	0.233	0.227
13	Default	1,2,2,3	0.398	0.393
14	$s_{12} = \frac{2}{3}$, $s_{22} = 2$	1,2,2,3	0.302	0.308
15	$s_{12} = 0.5$, $s_{22} = 0.6667$ $s_{32} = 2.$	1,2,2,3	0.332	0.304
16	$s_{12} = 0.5$, $s_{22} = 0.8$ $s_{32} = 1.2$ $s_{42} = 1.5$	1,2,2,3	0.384	0.378
17	$s_{12} = 1.5$, $s_{22} = 1.2$ $s_{32} = 0.8$ $s_{42} = 0.5$	1,2,2,3	0.368	0.374
18	$s_{12} = 0.5$, $s_{22} = 1.5$ $s_{32} = 1.5$ $s_{42} = 0.5$	1,2,2,3	0.309	0.308

4 Task Pipelines with Branching Flows

With flow branching there can be any number of parallel streams of pipeline tasks, as well as the features of the previous two sections.

Figure 4 shows an example. The joining of parallel flows is excluded from this study because it raises a distinct set of modelling problems which require special treatment, for instance in the split-join queue analysis of Flatto and Hahn [FLAT84].

The durations of phases 1 and 2 are the same as before, apart from differences in W described later. However the rest of the task is affected by splitting of the flow, for a task may now have several successors in the pipeline, and there will be a separate phase providing a RNV with each one (and a delay w) to pass on the data. Task j is a successor of task i if there is a phase p with $y_{ijp} = 1$, $p \in (3, P_i)$, where P_i is the number of successors of task i, plus 2. Without loss of generality the tasks are numbered so that data is passed only from lower-numbered tasks to higher-numbered tasks.

The phase durations are now:

$$x_{i1} = s_{i1} + (1 - Prob\{E_i\})W_{iE}, \quad x_{i2} = s_{i2}, \quad i \in (1, N) \tag{20}$$

$$x_{ip} = \sum_{j=i+1}^{N} y_{ijp}(w_{ij} + Processor\_waiting), \quad i \in (1, N),\ p \in (3, P_i) \tag{21}$$

$$w_{ij} = x_{j1} + \sum_{p=2}^{P_j} \text{Prob}\{E_{jp}\}(V_{jp} + \sum_{u=p+1}^{P_j} x_{ju}), \quad i, j \in (1, N) \tag{22}$$

$$Prob\{E_{jp}\} = (1 - Prob\{E_{j2}\}) \ldots (1 - Prob\{E_{j,p-1}\}) x_{jp} / (x_{jp} + (1/f) - w_{ij}) \tag{23}$$

$$V_{jp} = MRL(\tilde{x}_{jp}) \tag{24}$$

There are two quantities in (20)-(24) which are affected by splitting of flows, *(Processor_waiting)* in x_{ip}, and V_{jp}; consider the former first. With more than one RNV, the probability π_{ip} of seeing contention set K_{iE} when task i is ready depends on the phase p in which the RNV takes place. As time goes on, the probability increases, as the sender task is progressively more likely to have finished. Thus in (21) the term *(Processor_waiting)* is given by the following (in place of (9)):

$$Processor\_waiting = (1 - \pi_{ip}) W_{i\bar{E}} + \pi_{ip} W_{iE} \tag{25}$$

The value of π_{ip} is:

$$\pi_{ip} = Prob\{E_{i2}\} + \ldots + Prob\{E_{ip}\} \tag{26}$$

V_{jp} is similar to (19), with task k being the successor task sent to by phase p when there are several successors at a split:

$$\begin{aligned} V_{j2} &= \quad MRL\{\tilde{s}_{j2}\} \\ V_{jp} &= \quad MRL\{\tilde{w}_{jk} + \tilde{W}_{jE}\}, \;\; \text{for } p = 3, \ldots P_j \\ & \qquad j = 1, \ldots, N \end{aligned} \tag{27}$$

The processor-waiting calculation is the same as in Section 3, except that the task which sends a message to task i (there is still only one) is not necessarily task $i-1$. Thus the contention set definition (7) generalizes to:

$$K_{iE} = \{k | J(k) = J(i); \;\; k \neq i; \;\; \sum_p y_{kip} = 0\} \tag{28}$$

The computation of *Pwait(1)* by (14) follows directly, and of *Pwait(2)* by (15) also, when (17) and (18) are modified appropriately. In both (17) and (18) the expression on the right side before the fraction generalizes to

$$\begin{aligned} \mathrm{Prob}\{\textit{task k in the queue}\} = f[(1 - \mathrm{Prob}\{E_k\}) W_{kE} \\ \sum_{p=3}^{P_k} \sum_j y_{kjp}((1 - \pi_{kp}) W_{k\bar{E}} + \pi_{kp} W_{kE})] \end{aligned} \tag{29}$$

This completely determines the W terms.

These component calculations are now assembled into the following algorithm.

Algorithm for Complex Pipeline Throughputs

Step 1: Initialize $f = 1/(max_i(s_{i1} + s_{i2}))$, V_{N4} from (4), and $x_{i1} = s_{i1}$, $x_{i2} = s_{i2}$, $x_{ip} = 0$, $(p = 3, P_i)$, for all $i \in (1, N)$.

Step 2: For each j in the order $j = N-1, N-2, ...1$ determine:

(a) w_{jk} from (22), with (5), (27) for all $k \in (j+1, N)$;

(b) W_{jE} from (8), with (28), (12), (14), (15), and (17) as modified by (29);

Table 3: Complex Pipeline Throughputs

Six tasks as in Figure 4(a) with default parameters: $s_{i1} = 0.1$, $s_{i2} = 1.0$, all $i = 1,..6$ and $y_{123} = y_{233} = y_{235} = y_{343} = y_{453} = 1$. The processor allocation is given by a list $J(1), J(2), J(3), J(4), J(5), J(6)$.

Case	Parameters (differences from defaults)	Processor Allocation	Exact Throughput	Approximate Throughput
1	Default	1,2,3,4,5,6	0.421	0.389
2	$s_{12} = 0.667,\ s_{22} = 2$	1,2,3,4,5,6	0.337	0.315
3	$s_{12} = 0.5,\ s_{22} = 0.667\ s_{32} = 2.\ s_{52} = 2.$	1,2,3,4,5,6	0.310	0.292
4	$s_{62} = 2.$	1,2,3,4,5,6	0.353	0.336
5	$s_{12} = 0.3,\ s_{22} = 0.7\ s_{42} = 1.3\ s_{62} = 1.7$	1,2,3,4,5,6	0.392	0.371
6	$s_{12} = 1.7,\ s_{22} = 1.3\ s_{42} = 0.7\ s_{62} = 0.3$	1,2,3,4,5,6	0.354	0.343
7	Default	1,1,2,2,3,3	0.312	0.307
8	$s_{12} = 0.667,\ s_{22} = 2$	1,1,2,2,3,3	0.268	0.252
9	$s_{12} = 0.5,\ s_{22} = 0.667\ s_{32} = 2.\ s_{52} = 2.$	1,1,2,2,3,3	0.247	0.258
10	$s_{62} = 2.$	1,1,2,2,3,3	0.258	0.248
11	$s_{12} = 0.3,\ s_{22} = 0.7\ s_{42} = 1.3\ s_{62} = 1.7$	1,1,2,2,3,3	0.291	0.273
12	$s_{12} = 1.7,\ s_{22} = 1.3\ s_{42} = 0.7\ s_{62} = 0.3$	1,1,2,2,3,3	0.281	0.271
13	Default	1,1,1,1,1,1	0.154	0.143
14	$s_{12} = 0.667,\ s_{22} = 2$	1,1,1,1,1,1	0.140	0.128
15	$s_{12} = 0.5,\ s_{22} = 0.667\ s_{32} = 2.\ s_{52} = 2.$	1,1,1,1,1,1	0.130	0.113
16	$s_{62} = 2.$	1,1,1,1,1,1	0.133	0.119
17	Default	1,1,1,1,2,2	0.222	0.211
18	$s_{12} = 0.667,\ s_{22} = 2$	1,1,1,1,2,2	0.194	0.181
19	$s_{12} = 0.5,\ s_{22} = 0.667\ s_{32} = 2.\ s_{52} = 2.$	1,1,1,1,2,2	0.196	0.183
20	$s_{62} = 2.$	1,1,1,1,2,2	0.203	0.195
21	$s_{12} = 0.3,\ s_{22} = 0.7\ s_{42} = 1.3\ s_{62} = 1.7$	1,1,1,1,2,2	0.236	0.216
22	$s_{12} = 1.7,\ s_{22} = 1.3\ s_{42} = 0.7\ s_{62} = 0.3$	1,1,1,1,2,2	0.197	0.185

(c) $W_{j\bar{E}}$ from (8) with (22), equations symmetrical with (12), (14), (15) by interchanging E and $\bar{E}$, and (18) as modified by (29);

(d) x_{jp} from (20), (21) with (22), (23), (24).

Step 3: set $f = 1/(x_{12} + x_{13})$. If within a preset percentage of the previous value, stop, else repeat from step 2.

Examples and Accuracy

An example system is shown in Figure 4(b), with six tasks and one branching point. Table 3 shows some results of using the full algorithm on this case.

The approximation shows a tendency to underestimate the throughput by around 6%, a larger error here than without the splitting of flow.

The mean error is -5.7% and the RMS error is 6.6%. The worst error occurred in case 16 with all tasks allocated to a single processor, which is odd. Cases with the execution bottleneck at the end of the pipeline, such as 4, 5, 10, 11, 15, 16, 21 showed a tendency to larger percentage errors; in these cases the effect of delay at the end is pushed back through the waiting calculations to the entry throughput, and errors have more opportunity to accumulate. But case 20 is similar, with a small error.

In an iterative algorithm convergence and computational complexity must be considered. Convergence was always obtained in a few steps, but a formal proof of convergence has not been found. It was necessary to modify the iteration update step to apply only a fraction (50% was used) of the calculated change in waiting values w_{ij} at each step. Time complexity is of the order of N^3, the same as in [WOOD88], and space complexity is also similar at N^2.

5 Conclusions

The various possible structures of contention for resources in a software pipeline using rendezvous communication have been studied in detail, including contention for processing resources and shared software resources. Approximations were found within a new framework for estimating queueing delays, called "rendezvous networks". Significant extensions to the rendezvous network approach were made in the process. The resulting analysis was extensively tested on examples and gives throughput accuracies from 1% to nearly 15%, but almost all under 10%.

Some direct generalizations can be described. Software pipelines often include shared server tasks with access from various points in the pipeline, as illustrated in Figure 4(b), but space limitations require that models for these cases be described elsewhere.

The mean delay for one path in a pipeline is easily calculated as the sum of phase durations:

$$Pipeline\ delay \ = \ x_{11} + \sum_{i=1}^{N}(x_{i2} + x_{i3})$$

The first phase of each task after the input is not included in the sum because it forms part of the preceding phase 3.

For cases with Poisson arrivals at a fixed rate ("open" models) the throughput is known and it is delays that are of interest. One pass through the algorithm determines the phase durations and MRLs; then the input queue waiting time is found by considering only the first task as a server to the arrival stream. It is modelled as a special server, termed an M/G+G/1 server in [LENY88]. The result of Skinner [SKIN67] gives the mean input queue waiting time as:

$$Input\ wait \ = \ fExp(x_1^2)/2(1 - f\sum_p x_{1p})$$

where $Exp(x_1^2)$ is the mean square of the total service time of task 1, given by:

$$Exp(x_1^2) = (\sum_p x_{1p})^2 + Var(\sum_p \tilde{x}_{1p})$$

Because the phases are independent the variances add:

$$Exp\{x_1^2\} = (\sum_p x_{1p})^2 + \sum_p Var(\tilde{x}_{1p})$$

Total delay from arrival of a unit of input, to its completion, can be found by combining the input delay and the pipeline delay calculations.

6 References

[AJMO84] 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*, v 2 n 2 May 1984.

[CHER83] D. R. Cheriton, W. Zwaenpol, "The distributed V-kernel and its performance for diskless workstations", *Proc. 9th Symp on Operating Systems Principles*, ACM Operating Systems Review v17 n 5, pp 129-140, Oct 1983.

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

[FLAT84] L. Flatto, S. Hahn, "Two parallel queues created by arrivals with two demands", *SIAM J. on Appl. Math.*, v 44, n 5, 0ct. 1984.

[GAMM84] N. D. Gammage, L. M. Casey, "The software architecture of a distributed processing system", *Proc. 4th Int Conf. on Distributed Computing Systems*, IEEE Catalog 84CH2021-4, pp 414-431, May 1984.

[JACO82] P. A. Jacobson, E. D. Lazowska, "Analyzing Queueing Networks with Simuilataneous Resource Possession", *Comm. ACM.*, v 25, no 2, Feb. 1982.

[KANT87] K. Kant, "Modelling Interprocess Communication in Distributed Programs", *Proc. Int. Wkshop on Petri Nets and Performance Models*, Madison, Wisc., Aug. 1987, IEEE Catalog 87TH0815-9, pp 75-83.

[KLEI75] L. Kleinrock, *Queueing Systems*, v 1, Wiley, 1975.

[KOGG81] P. M. Kogge, *The Architecture of Pipelined Computers*, Hemisphere, Washington, 1981.

[LENY88] L. M. Le Ny, C. M. Woodside, "Performance Modelling of Queues with Rendezvous Service", Report OCIEE-88-01, Dept of Systems and Computer Engineering, Carleton University, Ottawa, Canada, April 1988.

[REIS79] M. Reiser, "A queueing analysis of computer communication networks with window flow control", *IEEE Trans. on Communications*, v COM-27, pp 1199-1209, Aug. 1979.

[SCHA88] S. M. Schatz and W.K. Cheng, "A Petri Net framework for automated static analysis of Ada tasking behaviour", to appear in *J. Systems and Software*, 1988.

[SKIN67] C. Skinner, "A priority Queueing System with Server Walking Time", *Operations Research*, v 15 pp. 278-285, 1967.

[WOOD84] C. M. Woodside, E. Neron, E. D-S. Ho, B. Mondoux, "An "active server" model for the performance of parallel programs written using rendezvous", *Preprints of Int. Workshop on Modelling and Performance Evaluation of Parallel Systems*, pp 157-174, Grenoble, Dec. 1984. Also published in *J. Systems and Software*, v6 nos 1 and 2, pp 125-132, May 1986.

[WOOD88] C. M. Woodside, "Throughput Calculation for Basic Stochastic Rendezvous Networks"; to appear in *Performance Evaluation*, late 1988.

Appendix: Mean Residual Life

We consider the mean residual life V of a summation variable $\tilde{X}$ with mean X:

$$\tilde{X} = \tilde{s} + \sum_{i=1}^{M} \tilde{z}_i$$

where $\tilde{s}$ has a general distribution with mean s, $\tilde{z}_i$ has an exponential distribution with mean z_i, M is a constant and all the components of $\tilde{X}$ are independent of each other. It is well known (see e.g. [KLEI75]) that

$$V = Exp\{\tilde{X}^2\}/2Exp\{X\}$$
$$Exp\{z_i^2\} = 2z_i^2$$

Thus:

$$V = (Exp\{(\tilde{s} + \sum_i^M \tilde{z}_i)^2\})/2X = (Exp\{\tilde{s}^2\} + 2s\sum_i z_i + Exp\{(\sum_i z_i)^2\})/2X$$
$$= (Exp\{\tilde{s}^2\} + 2s\sum_i z_i + \sum_i z_i^2 + (\sum_i z_i)^2)/2X$$

If all the components are exponentially distributed we can set $s = 0$ in the above.