Performance Bounds for Concurrent Software with Rendezvous

S. Majumdar C.M. Woodside J.E. Neilson
D.C. Petriu

Telecommunications Research Institute of Ontario
Department of Systems and Computer Engineering, and
School of Computer Science
Carleton University
Ottawa K1S 5B6, Canada

February 11, 1998

Abstract

Synchronous message-passing communication, or rendezvous, occurring between software tasks can have a significant effect on system performance. The rendezvous style of communication is coming into wider use in programming languages and operating systems for parallel and distributed environments. Understanding the performance implications of this style of inter-task communication is becoming a matter of practical importance. The dual nature of a task which acts both like a customer as well as a server, makes the performance analysis of rendezvous-based multitasking systems quite different from the analysis of the other queueing systems with known results. This research focuses on rendezvous-based systems in which the execution behavior of the software has a non-deterministic component of a very general nature which may for example be the manifestation of a data dependent behavior. Based on a model called the Stochastic Rendezvous Network the computation of bounds on task throughputs for multitasking systems characterized by rendezvous style communication is presented. Although the behavior of tasks is called stochastic, it is very general and the results are valid for general distributions of computation times and the number of messages generated by tasks. The inter-relationship among task throughputs, however, makes it difficult to extract the bounds in closed analytic form. The notion of a feasible throughput region which encloses the set of feasible task throughputs and captures the inter-relationship among the behavior of tasks is introduced. Variations of this basic bounding approach that are useful in the context of different types of multitasking systems are considered in the paper. For example, a novel technique based on interval arithmetic is proposed for the computation of numerical values for the bounds. The applicability of the bounds and their tightness are analyzed through case studies. Issues such as the inter-relationship between the software architecture and system performance, and the effect of processor contention on the performance bounds are discussed.

Key Words: Distributed Software, Message Passing, Multitasking Systems, Performance Evaluation, Rendezvous, Software Performance.

1 Introduction

Concurrent systems characterized by communicating tasks have been a major focus of attention of current research in distributed and parallel systems. This paper is concerned with an important class of concurrent systems that are comprised of parallel tasks using *rendezvous* for inter-task communication. A task in such a system represents a unit of work with a sequential thread of control. Inter-communication between tasks is synchronous in the sense that the task requesting the rendezvous sends a message to another task and waits until it receives a reply from the receiver task. If the receiver task was already blocked on a "receive" primitive it starts processing the message immediately. Otherwise the message processing is delayed by the interval of time taken by the receiver task to reach the rendezvous point. If several tasks send messages to the same task simultaneously the messages are queued at the receiver task and are served according to a predefined queueing discipline. After processing a message and issuing a reply, the receiver goes into a post-rendezvous phase of operation concurrently with the sender task which resumes processing after receiving the reply to the message.

The rendezvous feature is used in a large number of programming languages and operating systems that support concurrency. Examples include Occam [2], the Thoth operating system [9] and the V-System [10] in which send-and-wait and receive-and-wait primitives are used for inter-task communication. Ada [8], and XMS [20] use procedural request/accept constructs which are equivalent to the rendezvous concept from the control flow point of view. Remote Procedure Calls supported by the SUNOS and the Mach operating system [22] can also be modelled as special cases of rendezvous-based intercommunication in which there is no post-rendezvous phase of execution. Modeling of rendezvous systems is vital both for performance evaluation of rendezvous-based systems as well as for understanding their behavior.

The modelling of concurrent systems has received considerable attention from researchers in the field. Only a representative set of this work is discussed here. A task in a rendezvous-based system behaves both as a server (by receiving messages) as well as a customer (by sending messages to other tasks). This dual nature of tasks makes it difficult to use standard queueing network models [11] for the performance analysis of concurrent interacting task systems. An approximate analysis of inter-communicating task systems based on the MVA technique is presented in [13]. Although the model presented in [13] is applicable to rendezvous-based systems, the post-rendezvous phase of task execution is not considered. Standard queueing theory does not deal with models of parallel execution with inter-dependence between customers. A queueing theoretic approach has been used in [12] for the performance analysis of a limited variety of splits and joins in control flow but this model does not extend to general systems of concurrent tasks using a rendezvous mechanism for interaction. A direct derivation of a Markov chain model of a system characterized by tasks communicating through non-blocking message passing is proposed in [24]. A key feature of rendezvous-based systems, however, is the blocking of a task when it goes into a rendezvous with another task. Petri nets [21] can model arbitrary interaction among parallel activities and the incorporation of timed transitions and queues have made the technique amenable to performance modelling. Holliday and Vernon describe a Petri net-like modelling framework which leads to a Markov chain performance model of the system [18]. A variant of this model and its use in modelling interacting task systems is reported in [16]. Analysis of a Petri Net model for producer-consumer task systems in which a single consumer task interacts with multiple producer tasks through rendezvous is presented in [7]. For inter-communicating task systems in general, Petri net-based approaches can be used for performance evaluation as long as the number of tasks in the system is small. With an increase in the number of tasks the system state space grows very fast making the Petri net-based approach ineffective.

Stochastic behavior of software has been the focus of attention in numerous analytical studies some of which were included in the discussion provided in the previous paragraph. Non-deterministic behavior of program execution is also captured through measurements made on real systems. For example, a study [25] of the overhead due to network communication for a multiuser system running under 4.3 BSD UNIX demonstrates the variability in the execution times of the interrupt service routine from the service of one network event to another. The research presented in this paper is concerned with rendezvous-based systems that are stochastic in nature. Studying such systems is important when the execution behavior of the software is non-deterministic and may depend on the set of input data or real time events for example.

A *Stochastic Rendezvous Network (SRVN)* model is proposed in [3]. The author uses an implicit decomposition approach to obtain approximate values for the throughput of tasks in a rendezvous-based systems. The technique is efficient even for systems with a large number of interacting tasks and is observed to produce a reasonable degree of accuracy in estimating the system throughput. The analysis of the SRVN model has been enhanced in [14] to handle concurrent systems in which priorities can be associted with the component tasks, in [15] to capture the effect of processor contention when the executing tasks in the system compete for a limited number of processors, and in [4] to handle systems of tasks with specific ports or entries. The SRVN modelling technique has also been used to analyze specific systems, such as a system characterized by a complex software pipeline [5].

The characteristics of concurrent software can have a major impact on the overall system performance. For example, the performance bottleneck for the system may be a software element rather than a hardware device [19]. A major strength of the SRVN is its capability to model both the software as well as the hardware components of the system. Techniques for estimating the system performance from these models are important for understanding the impact of the characteristics of the system components on performance. This paper discusses the development of throughput bounds for rendezvous-based systems modelled by SRVNs. The bounds are applicable to SRVNs in which each task has a single entry so that the behaviour of a task receiving a rendezvous message is independent of the identity of the sender task. These bounds are useful not only for performance evaluation of rendezvous-based systems but also for obtaining valuable insights into system behavior in terms of the important factors affecting system performance. The exact analysis of the SRVN model through Petri nets as well as the approximate analysis presented in [3] are based on assumptions about the distribution of task execution times. The bounds presented in this paper, however, are valid for general distributions of task computation times and the number of messages generated by tasks; only an independence assumption and the use of a FIFO (First-In-First-Out) queueing discipline at tasks for servicing rendezvous messages are required for the existence of the bounds (see Section 2 for a specification of the assumptions that underly the bound computation). These bounds are thus robust and applicable to a wide variety of rendezvous-based systems. Although the approximate solution of the SRVN model presented in [3] has been found to be effective for a large number of rendezvous-based systems, there is no guarantee on the magnitude of the error produced in the estimation of task throughputs for any given arbitrary SRVN. The exact values for these throughputs are guaranteed to lie within the interval specified by the bounds that this paper focuses on. An important aspect of these bounds is their ability to capture the inter-relationship among the behavior of tasks in the multi-tasking system. The notion of a feasible throughput region which encloses the set of feasible throughput values and the graphical presentation of the region demonstrating the inter-dependence among task throughputs are discussed in the paper. This inter-dependence among task throughputs, however, makes it difficult to extract the bounds in closed analytic form. A technique based on interval arithmetic for the computation of numeric values for the bounds is presented.

A description of the SRVN model is presented in Section 2. Bounds on the performance of SRVN systems are derived in Section 3. Section 4 discusses the computation of these bounds and their presentation to the system designer. The tightness of the bounds and its relationship with the system parameters are discussed in the context of two generic software architectures in Section 5. A comparison between the bounds for a client-server architecture based on the SRVN model and those obtained from an approximate queueing network model of the system is also presented in Section 5. Section 6 presents the incorporation of hardware resources such as processors into the model and discusses the effect of processor contention on the tightness of bounds. Our conclusions are presented in Section 7.

2 The Stochastic Rendezvous Network Model

If the structure of a program with several inter-communicating tasks is examined, a graph that we call the SRVN graph can be extracted. Two example SRVN graphs are shown in Figure 1. The nodes labelled $1 \ldots N$ of the graph correspond to tasks, and there is a directed edge (i, j) if task i sends a rendezvous message to task j. We will restrict our attention to *acyclic* graphs since cycles may cause mutual waiting or "rendezvous deadlock". Then without loss of generality the tasks in the SRVN graph can be labelled such that $i < j$ for every edge (i, j).

Each task in the SRVN behaves in a cyclic fashion and a cycle of operation is characterized by two distinct phases, p ($p = 1, 2$), of execution (see Figure 2). During any phase of execution a task performs computations and may (optionally) send messages to other tasks in the system. The first phase or phase-1 is special in that it corresponds to a rendezvous operation and is called the in-rendezvous phase. The first phase of execution for task j is initiated by the reception of a message from another task i; in response to the message task j performs the necessary operations and sends back a reply to the calling task. Until the reply is received the calling task i remains blocked. If task j is busy when the rendezvous call is made by i, task i joins a FIFO queue. At the end of its first phase of execution for servicing a message from i, task j sends a reply back to task i (which can resume execution) and goes into its second or post-rendezvous phase of execution. If there is no task waiting for a rendezvous, task j blocks on a receive primitive at the end of its current cycle.

There can be two types of tasks in an SRVN: *reference tasks* and *non-reference tasks.* Typical operations performed by each type of tasks are shown in Figure 3. The reference tasks function as drivers of the system and have no first phases; they send messages to other tasks but receive none. Looping endlessly the reference tasks use the services of other non-reference tasks through rendezvous calls. The set of reference tasks in the system is denoted by $\mathcal{R}$ and without loss of generality the nodes corresponding to the reference tasks are labeled $1 \ldots R$ in the SRVN graph. The remainder of the tasks in the SRVN constitute the set $\mathcal{N}$ of non-reference tasks that are labelled $R + 1 \ldots N$. The non-reference tasks receive as well as send messages to other tasks in the system. The set of tasks to which a task labelled $i \in \mathcal{R} \cup \mathcal{N}$ sends messages is denoted by $\mathcal{N}_i$ (see Figure 1(a) for example).

A rendezvous may be viewed as being initiated by a reference task. A rendezvous between a task m and a task j is called direct when there is an edge between m and j, and is called nested when j is invoked by m through a chain of other non-reference tasks. In general a *path* is said to exist between task m and any other task that is activated by m. A path between task m and a task k consists of a particular sequence of tasks that participate in the invocation of task k by m. The path is labelled with a string that begins with task label m and contains a series of task labels ordered in the sequence in which these tasks are activated. For example, if task m calls task j which in turn calls task k then there is a path between tasks m and j as well as between tasks m and k. The first path is labelled mj while the second is labelled mjk.

The input parameters for the model define the characteristics of the different phases of execution for each task in the system. Typically the duration of a phase is the sum of the computation times for the task and the time spent in performing rendezvous with other tasks. Each phase p for a task j is characterized by the mean computation time and the mean number of rendezvous requests made to other tasks in the system. The mean computation time is denoted by s_{jp} and the mean number of rendezvous requests made to a task k ($k \in \mathcal{N}_j$) is denoted by y_{jkp}. The mean computation time per cycle (over both phases) is represented by $S_j = (s_{j1} + s_{j2})$ and the mean number of rendezvous with k per cycle by $Y_{jk} = (y_{jk1} + y_{jk2})$. The delay introduced by the communication medium is assumed to be negligibly small. The research presented in this paper concentrates on the effect of the behavior of the software as well as the limitation in processing resources on system performance and the results are applicable primarily to systems in which the communication medium is not a critical resource. It is possible however, to approximate the delay introduced by the communication medium by appropriately inflating the computation time for the first phase of execution of each non-reference task.

Note that only the first moments for the input parameters, $s_{j1}, s_{j2}, y_{jk1}, y_{jk2}$, are required for the computation of the bounds. The bounds do not depend on the nature of the distributions underlying the task computation times and the number of messages generated by tasks. Only the assumptions that the task computation times during any phase are independent with any general IFR (Increasing Failure Rate) distribution and the number of rendezvous requests generated by tasks during any phase are independent random variables are required for the existence of the bounds.

The performance measures of interest are the throughput of the reference tasks, f_r ($r \in \mathcal{R}$). f_r is the rate at which task r cycles. The throughput of each non-reference task j ($j \in \mathcal{N}$) depends on the rate of receiving messages from other tasks and is given by:

$$f_j = \sum_{i<j} f_i Y_{ij}$$

The above throughput equation can be simplified (for example by Gaussian elimination) and expressed in terms of the throughputs and message rates of reference tasks:

$$f_j = \sum_{r \in \mathcal{R}} a_{jr} f_r \tag{1}$$

Note that for a task j, the coefficient a_{jr} is the sum of the mean number of direct rendezvous with j initiated per cycle by r and the mean numbers of nested rendezvous initiated by r, per cycle, through all the paths between r and j.

3 Bounds on Throughputs of Reference Tasks

The reference tasks $r \in \mathcal{R}$ are the drivers of the system and the set of throughputs of these reference tasks is a fundamental measure for system performance. The throughputs of the non-reference tasks can be expressed in terms of the reference task throughputs (see equation 1). Our goal is to compute a set of bounds on expected reference task throughputs:

$$f_r^+ \geq f_r \geq f_r^- \text{ forall } r \in \mathcal{R}$$

or more general bounding functions of the form $G(f_r) \leq 0$ that express the constraints on f_r (for $\forall r \in \mathcal{R}$).
Unless mentioned otherwise all the parameters used in the derivation of these bounds are expected values of random variables.

Upper bounds on the throughput of reference tasks were discussed in [3]. This paper focuses primarily on the derivation of the lower bound on the throughputs of reference tasks, and the application of these upper and lower bounds for performance prediction as well as for gaining insight into the behavior of the multitasking system. A brief discussion of the upper bounds on throughput is presented in this section which is followed by the analysis of lower bounds. Each task is assumed to be executing on its own processor. This restriction is removed in Section 6 which considers the incorporation of processor contention into the model.

The computations of the bounds on reference task throughputs are based on the analysis of their cycle times. The cycle time C_r of reference task r consists of its own computation time and the delays introduced by the non-reference tasks in the system for the service of messages initiated by r. The delay introduced by a non-reference task k may depend on the path through which the message arrives at k. This is because the queueing delay experienced by a message at task k is dependent on the path through which the message arrives. Consider for example the SRVN shown in Figure 1(a). There are two paths between task 1 and task 6. A message for task 6 coming through path 16 may have to wait for the completion of service for two messages, one from task 4 and the other from task 5. On the other hand a nested rendezvous message arriving at task 6 through path 146 may have to contend with only one message which is initiated by task 5. In order to capture these different delay components which contribute to C_r we analyze the different paths between two tasks. The set of distinct paths between two tasks m and k is denoted by $\Pi(m, k)$. With each path $\pi \in \Pi(m, k)$ we associate a delay which is defined as follows.

d_π = time taken by task k to service a message arriving through path π.

> [During the interval d_π all the tasks that precede task k in the path π remain blocked. Task k may be executing during this interval or may itself be blocked awaiting replies from other tasks].

The analyses of these delay components (d_π) form an important component of the discussion presented in this section.

3.1 Upper Bounds

In this sub-section the upper bound results of [3] are recast in a way that introduces the analysis of the lower bound on task throughputs which is presented in the following sub-section. The upper bounds on throughput are of two types. The first type of bounds is called *no contention* bounds and are computed by ignoring the queueing delay at a non-reference task for the service of a message. The second type of bounds is called *utilization-based* bounds and are based on the constraint that the utilization of a task cannot be more than 1. Computation of each of these bounds for the SRVN model is presented.

In order to compute f_r^+ for any $r \in \mathcal{R}$, a lower bound C_r^- on the cycle time of the reference task needs to be computed: $f_r^+ = 1/C_r^-$. The cycle time of reference task r consists of its own computation time S_r, and the time during which it remains blocked in performing rendezvous with other non-reference tasks $j \in \mathcal{N}_r$.

$$C_r = S_r + \sum_{j \in \mathcal{N}_r} Y_{rj} d_{rj}$$

A lower bound $d_{rj}^- \leq d_{ij}$ will now be defined, so that:

$$C_r^- = S_r + \sum_{j \in \mathcal{N}_r} Y_{rj} d_{rj}^- \tag{2}$$

The delay introduced at j in the service of a message corresponds to task j's first phase of execution and consists of its own computation time s_{j1} and the time during which it remains blocked in performing rendezvous with other non-reference tasks $k \in \mathcal{N}_j$. If j is a *pure server* task and does not call any other task in the SRVN then $d_{rj}^- = s_{j1}$. In general, d_{rj}^- can be expressed as:

$$d_{rj}^- = s_{j1} + \sum_{k \in \mathcal{N}_j} y_{jk1} d_{rjk}^- \tag{3}$$

For any path π between tasks r and k, d_π^- is equal to the phase-1 computation time for task k plus all the nested rendezvous phase-1 computation times. Note d_π^- includes the phase-1 computation times of tasks only and not the queueing delays at tasks for rendezvous. Thus the d_{rjk}^-'s in the previous equation can be expanded in turn and the series of expansions can be continued until a pure server task is reached. Reference task r initiates rendezvous with different non-reference tasks in the SRVN, and every non-reference task $k \in \mathcal{N}$ that is activated by r contributes to C_r. By expanding the sum term in equation 2 and collecting all terms involving s_{k1} together we obtain after a little manipulation:

$$C_r^- = S_r + \sum_{k \in \mathcal{N}} a_{kr1} \; s_{k1}$$

which yields:

$$f_r \leq f_r^+ = \frac{1}{S_r + \sum_{k \in \mathcal{N}} a_{kr1} \; s_{k1}} \tag{4}$$

The a_{kr1} coefficients assigned in the expansion of equation 2 are found as follows. Reference task r may invoke task k through a number of different paths $\pi \in \Pi(r,k)$. The total number of times k is invoked by r in a single cycle through all the paths in $\Pi(r,k)$ is given by a_{kr} (see equation 1). As in the case of a_{kr}, all the paths in $\Pi(r,k)$ contribute to a_{kr1}. Since the no contention bound ignores the waiting at any task along a given path, only those invocations of k during which all the non-reference tasks in the path are blocked in their first phases contribute to a_{kr1}. Thus

$$a_{kr1} = Y_{rk} + \sum_{(r<j<k)} a_{jr1} \; y_{jk1} \tag{5}$$

By processing the tasks in increasing order of task labels a_{kr1} for any $k \in \mathcal{N}$ can be computed.

The second type of bounds are based on the fact that the utilization of any task cannot exceed 1.0. Thus for any task $j \in \mathcal{N}$:

$$f_j^+ = \frac{1}{C_j^-}$$

Note that the lower bound on cycle time, C_j^-, for any non-reference task j can be computed in the same manner as described by equations 2 and 3 for reference task r:

$$C_j^- = S_j + \sum_{k \in \mathcal{N}_j} Y_{jk} d_{jk}^-$$

where d_{jk}^- for each $k \in \mathcal{N}_j$ can be expressed as the sum of s_{k1} and all the nested rendezvous phase-1 computation times. By using this equation in conjunction with equation 1 we get an inequality for every $j \in \mathcal{N}$:

$$\sum_{r \in \mathcal{R}} a_{jr} f_r = f_j \leq f_j^+ = \frac{1}{S_j + \sum_{k \in \mathcal{N}_j} Y_{jk} d_{jk}^-} \tag{6}$$

As in the case of equation 4 the previous equation can be reduced to the following form:

$$\sum_{r \in \mathcal{R}} a_{jr} f_r = f_j \leq f_j^+ = \frac{1}{S_j + \sum_{k \in \mathcal{N}} a_{kj1} \; s_{k1}} \tag{7}$$

where

$$a_{kj1} = Y_{jk} + \sum_{(j<n<k)} a_{nj1} \; y_{nk1} \tag{8}$$

The term a_{kj1} is similar to a_{kr1} defined in equation 5 except that a_{kj1} corresponds to the invocations of task k by the traffic produced by task j only.

3.2 Lower Bound

The lower bound f_r^- on throughput of a reference task r can be determined from an upper bound C_r^+ on its cycle time: $f_r^- = 1/C_r^+$. The cycle time of r consists of task r's own computation time S_r and delays when task r is blocked, awaiting replies to its messages. There are two sources for the delay introduced by a non-reference task: the delay introduced by the traffic from reference task r, and the delay due to the interfering traffic arriving at the non-reference task from all the other reference tasks in the system. The delay component due to the traffic from reference task r itself is computed in the next section which focuses on SRVNs with only a single reference task r. Then the interference from the other reference tasks in the SRVN is analyzed and is captured in a queueing bound that forms the core of the computation. Finally the contributions of the traffic from the reference task r and the interference from the other reference tasks in the SRVN are combined to obtain a general expression for C_r^+. Because the service time distributions are IFR, any mean residual service time is bounded above by the mean service time [17]. This is exploited several times in the following development by using the mean service time as a bound on the mean residual service time.

Single Reference Task

An SRVN with a single reference task r (see Figure 1(b)) is analyzed. For such an SRVN, a simple upper bound C_r^+ corresponds to the sequential execution of the multitasking system, with all parallelism removed:

$$C_r^+ = S_r + \sum_{k \in \mathcal{N}} a_{kr} S_k$$

Although it may be clear in the context of an SRVN with a single reference task that a sequential execution of the task system represents the most pessimistic scenario, a discussion of how this bound is achieved is presented for the purpose of developing the general methodology used for the analysis of more complex SRVNs.

Reference task r performs rendezvous with every task $j \in \mathcal{N}_r$, and its cycle time C_r is given by:

$$C_r = S_r + \sum_{j \in N_r} Y_{rj} d_{rj}$$

Note that from the independence assumptions (presented in Section 2) for computation times and the number of messages generated by a task, the mean included service from a non reference task is given by the product of $Y_r j$ and d_{rj}.

d_{rj} in the previous equation is the mean delay experienced per rendezvous call by r at task j and is given by:

$$d_{rj} = s_{j1} + \sum_{k \in \mathcal{N}_j} y_{jk1} d_{rjk} + I_{rj} \tag{9}$$

The first component of d_{rj} in equation 9 is the phase-1 computation time of task j, whereas the second component is the delay introduced by the tasks that are called by j in phase-1 for completing the service of a message from r. The third component I_{rj} is the delay introduced in the service of a rendezvous from r by the interference from the phase-2 execution of j corresponding to its previous invocation by reference task r. In order to obtain an upper bound $C_r^+ \geq C_r$, an upper bound $d_{rj}^+ \geq d_{rj}$ will be found to yield:

$$C_r^+ = S_r + \sum_{j \in N_r} Y_{rj} d_{rj}^+ \tag{10}$$

d_{rj}^+ in turn is found from 9 as:

$$d_{rj}^+ = s_{j1} + \sum_{k \in \mathcal{N}_j} y_{jk1} d_{rjk}^+ + I_{rj}^+ \tag{11}$$

where, d_{rjk}^+ is a bound on d_{rjk} and I_{rj}^+ is a bound on I_{rj} given by:

$$I_{rj} \leq I_{rj}^+ = s_{j2} + \sum_{k \in \mathcal{N}_j} y_{jk2} d_{rjk}^+ \tag{12}$$

The first component of I_{rj}^+ is the phase-2 computation time for task j and the second component is the period during which j remains blocked in its second phase of execution for performing rendezvous with other non-reference tasks in the system. Using equation 11 in conjunction with equation 12 yields:

$$d_{rj}^+ = S_j + \sum_{k \in \mathcal{N}_j} Y_{jk} d_{rjk}^+ \tag{13}$$

If task j is a pure server task (so that $\mathcal{N}_j$ is empty) then $d_{rj}^+ = S_j$. Otherwise, by substituting equation 13 in equation 10 we expand C_r^+ to include another level of messaging.

$$C_r^+ = S_r + \sum_{j \in \mathcal{N}_r} Y_{rj} S_j + \sum_{j \in \mathcal{N}_r} \sum_{k \in \mathcal{N}_j} Y_{rj} Y_{jk} d_{rjk}^+$$

If every task j is a pure server task and does not call any other tasks in the SRVN then the third term of the right hand side of the previous equation is 0. Otherwise, the process of expansion can be continued for another round, by expressing:

$$d^{+}_{rjk} = s_{k1} + \sum_{l \in \mathcal{N}_k} y_{kl1} d^{+}_{rjkl} + I^{+}_{rjk} \tag{14}$$

where I^{+}_{rjk} is the upper bound on the delay introduced in the service of a message from j by the interference from the phase-2 execution of k due to a previous invocation by reference task r through task j. The sequence of expansions can be continued in this way until we reach a pure server task for every path that originates at r. Thus by expanding the sum term in equation 10 and collecting all the terms involving S_k together we finally obtain:

$$C^{+}_{r} = S_r + \sum_{k \in \mathcal{N}} a_{kr} S_k \tag{15}$$

where a_{kr} (see equation 1) is the number of times task k is invoked, through all the paths in $\Pi(r, k)$, in a single cycle of r. As noted above C^{+}_{r} is equal to the expected cycle time for a sequential execution of the task system modelled by the SRVN. Although this upper bound on cycle time, C^{+}_{r}, may be tight for some SRVNs it may be loose for some others. Consider for example the SRVN shown in Figure 1(b). The bound given by equation 15 is tight when the computation time for all tasks except task l is small and is achieved in the limiting case in which $S_m = 0$ for all $m \in \mathcal{R} \cup \mathcal{N};\ m \neq l$. Note that the derivation of C^{+}_{r} is based on the assumption that when a task j calls another task k, j always sees the entire second phase of execution of task k corresponding to its previous invocation. Thus the bound is potentially loose in situations in which there is a high degree of overlap between the computation of task j (given by S_j) and the phase-2 execution of task k. Considering such effects is quite complex in the context of general SRVNs and is not considered in this paper. As will be shown for general SRVNs with multiple reference tasks, the contribution of reference task r (as captured by equation 15) is only one component of C^{+}_{r}, which also includes queueing delays produced by the interference from other reference tasks in the system. The queueing delay introduced by each reference task is dependent on its throughput which makes C^{+}_{r} a function of the throughputs of all the other reference tasks in the SRVN. These queueing delays form a major component of the reference task cycle times and is discussed next.

Interference From Other Reference Tasks

The set of tasks with which reference task r directly performs rendezvous is denoted by $\mathcal{N}_r$. The delay experienced by r at each task $j \in \mathcal{N}_r$ has two components. The first corresponds to the traffic produced by the reference task r itself and was analyzed in the previous section. The second component corresponds to the interference produced by all other reference tasks $m \in \mathcal{R}$ $(m \neq r)$ in the SRVN. The computation of this second delay component is performed next. Let,

W_π = the expected delay experienced per cycle by reference task r for performing rendezvous with a given task k, through path $\pi \in \Pi(r, k)$, caused by the processing of messages originating from all other reference tasks $m \in \mathcal{R}$ $(m \neq r)$. Messages from a reference task m arriving at k through all the existing paths between m and k contribute to this delay.

Q_π = the expected number of messages, initiated by all other reference tasks, that delay the messages (initiated by r in one cycle) reaching task k through path $\pi \in \Pi(r, k)$.

Thus W_π is the expected delay experienced by task r for the service of Q_π messages at task k. Note that the delay introduced by the other reference tasks in the SRVN is computed for an entire cycle of r rather than on a per message basis. This enables us to capture more complex task interactions which in turn leads to a tighter bound.

By using equation 15 and introducing the interference from all the other non-reference tasks we get:

$$C_r \leq S_r + \sum_{k \in \mathcal{N}} a_{kr} S_k + \sum_{j \in \mathcal{N}_r} W_{rj} \tag{16}$$

Whenever r performs a rendezvous with j, it may find a number of messages initiated by other reference tasks enqueued in the FIFO queue at j. These messages are processed before the message from r can be serviced. Q_{rj} represents the total number of such messages that delay the messages sent by r directly to j in one cycle. In order to service a message from r, task j can perform rendezvous with other non-reference tasks in the system. During the processing of these nested rendezvous messages sent to $k \in \mathcal{N}_j$ both r and j remain blocked. Thus W_{rj} can be expressed as:

$$W_{rj} = Q_{rj} S_j + \sum_{k \in \mathcal{N}_j} W_{rjk} \tag{17}$$

If j is a pure server task and does not call any other task in the SRVN then W_{rjk} is 0, otherwise W_{rjk} can be expressed as:

$$W_{rjk} = Q_{rjk} S_k + \sum_{l \in \mathcal{N}_k} W_{rjkl} \tag{18}$$

By using equations 17 and 18 we get:

$$\sum_{j \in \mathcal{N}_r} W_{rj} = \sum_{j \in \mathcal{N}_r} Q_{rj} S_j + \sum_{j \in \mathcal{N}_r} \sum_{k \in \mathcal{N}_j} Q_{rjk} S_k + \sum_{j \in \mathcal{N}_r} \sum_{k \in \mathcal{N}_j} \sum_{l \in \mathcal{N}_k} W_{rjkl} \tag{19}$$

The series of expansions can be continued until a pure server task is reached for every path originating at task r.

In general, a reference task r may initiate the execution of a non-reference task k through a number of different paths. Each of these paths between tasks r and k contains a particular sequence of tasks that are activated by r through nested rendezvous. Thus by expanding the sum terms in equation 19 and collecting all the terms involving S_k together, we get:

$$\sum_{j \in \mathcal{N}_r} W_{rj} = \sum_{k \in \mathcal{N}} \sum_{\pi \in \Pi(r,k)} Q_\pi S_k$$

Thus every non-reference task k that is invoked by reference task r (directly or indirectly through nested rendezvous) contributes to $\sum_{j \in \mathcal{N}_r} W_{rj}$. For each such k, $\sum_{\pi \in \Pi(r,k)} Q_\pi$ denotes the expected number of interfering messages (per cycle of r) initiated by all other reference tasks and appearing at task k. During the execution of these messages r remains blocked. For a task $k \in \mathcal{N}$ that is not invoked by r, $\sum_{\pi \in \Pi(r,k)} Q_\pi = 0$.

If an upper bound $b_{kr} \geq \sum_{\pi \in \Pi(r,k)} Q_\pi$ can be computed then an upper bound $(\sum_{j \in \mathcal{N}_r} W_{rj})^+ \geq \sum_{j \in \mathcal{N}_r} W_{rj}$ can be obtained as:

$$(\sum_{j \in \mathcal{N}_r} W_{rj})^+ = \sum_{k \in \mathcal{N}} b_{kr} S_k \tag{20}$$

Note that for computing the upper bound we include the entire interval for all service intervals that can possibly conflict with the service of task r. Because the task computation times and the number of messages generated by tasks are independent and the inclusion of an execution interval of task k in $(\sum_{j \in \mathcal{N}_r} W_{rj})^+$ does not depend on its duration, the mean of these intervals included in the sum is the mean S_k.

Substituting equation 20 in equation 16 yields:

$$C_r \leq C_r^+ = S_r + \sum_{k \in \mathcal{N}} S_k(a_{kr} + b_{kr}) \tag{21}$$

The computation of b_{kr} will be presented next.

Computation of b_{kr}

The computation of b_{kr} is presented first in the context of a simple class of SRVNs in which all the non-reference tasks are pure servers. The analysis is then extended to general SRVNs. Both of these analyses are based on the following Lemma which captures the contention between two sources of messages. A message source may represent the entire or partial traffic from a task. For example, consider a task j in the SRVN that generates messages for another non-reference task k deeper in the graph. A message source can be used to represent either the total traffic from j to k or a part of the total traffic (messages initiated by a particular reference task for example) between j and k.

Contention may exist between message sources. If x and z are two message sources that contend with each other at task k then q_{kx}^z is defined as the expected number of messages from z that delay the messages arriving per second from x. Thus if λ_x is the message rate from x to k then these λ_x messages are expected to be delayed by the service of q_{kx}^z messages from z.

Lemma 1: Let x and z denote two sources of messages that perform rendezvous with task k. If λ_x and λ_z (messages per second) are the expected rates of messages that arrive at k from x and z respectively then:

$$q_{kx}^z \leq min(\lambda_x, \lambda_z)$$

Proof:

Case 1: $\lambda_z < \lambda_x$: Since λ_z is the total number of messages from z that arrive at k per second:

$$q_{kx}^z \leq \lambda_z$$

Case 2: $\lambda_z \geq \lambda_x$: Since the messages at k are served in FIFO order each message from x can be delayed by at most a single message from z. Thus,

$$q_{kx}^z \leq \lambda_x$$

From Case 1 and Case 2 it follows that:

$$q_{kx}^z \leq min(\lambda_x, \lambda_z)$$

QED

It can be easily verified that Lemma 1 holds even if λ_x or λ_z is an upper bound rather than the expectation for the message rate from source x or z.

Case (i) Simple SRVNs Without Nested Rendezvous

A simple situation in which non-reference tasks are pure servers (see Figure 4) and do not call any other task in the SRVN is considered first.

Reference task r is a source of message that generates messages for task k at the rate of $\lambda_r = a_{kr} f_r$. Every other reference task $m \in \mathcal{R}$ $(m \neq r)$ acts as a contending source of messages with rate $a_{km} f_m$. By applying Lemma 1 for r and each $m \in \mathcal{R}$ we have:

$$q_{kr}^m \leq min(a_{kr} f_r, a_{km} f_m)$$

Noting that q_{kr}^m / f_r is the expected number of messages from m that can delay a single cycle of r and summing over all the reference tasks $m \in \mathcal{R}$ $(m \neq r)$ we have:

$$b_{kr} = \sum_{m \in \mathcal{R};\, m \neq r} min(a_{kr}, \frac{a_{km} f_m}{f_r}) \tag{22}$$

Case (ii) General SRVNs with Nested Rendezvous

The essence of the bound on queueing delay captured by equation 22 is that a message can wait for at most one other message from each source. The previous analysis of b_{kr} can be extended to handle nested rendezvous at tasks k deeper in the graph (see Figure 5 for example), by using each *direct client* task j of k as an independent message source. A direct client task of k is a task in the SRVN that directly performs a rendezvous with k. The set of all the direct client tasks of k is denoted by $\mathcal{D}(k)$.

For the computation of b_{kr} the contribution of each task $j \in \mathcal{D}(k)$, b_{kr}^j, is computed first. b_{kr}^j is the upper bound on the number of messages to k (initiated by the other reference tasks) that delay a cycle of r when r calls k through task j. Note that these rendezvous between r and k through task j correspond to the messages that reach j through all the paths between r and j. Once b_{kr}^j is computed for each $j \in \mathcal{D}(k)$, b_{kr} is computed as:

$$b_{kr} = \sum_{j \in \mathcal{D}(k)} b_{kr}^j \tag{23}$$

For every direct client task j, $(a_{jr} + b_{jr})$ cycles of j are included in C_r^+. The a_{jr} cycles of j correspond to reference task r's own service and produce $a_{jr} Y_{jk}$ messages for task k. The b_{jr} cycles of j correspond to the interfering messages at task j from the other reference tasks $m \in \mathcal{R}$ $(m \neq r)$ and produce $b_{jr} Y_{jk}$ messages for task k. These $b_{jr} Y_{jk}$ messages form only one component of b_{kr}^j. The other component of b_{kr}^j is the upper bound on the number of messages at k that delay the $(a_{jr} + b_{jr}) Y_{jk}$ messages coming from j. In order to compute the second component of b_{kr}^j we represent the group of $(a_{jr} + b_{jr}) Y_{jk}$ calls by a message source x with rate $\lambda_x = (a_{jr} + b_{jr}) Y_{jk} f_r$. Each direct client task $l \in \mathcal{D}(k)$ $(l \neq r)$ is a source of contending messages that are initiated by the other reference tasks in the SRVN. We represent the contending messages for x forwarded by each

l by a message source z with rate:

$$\lambda_z = \sum_{m \in \mathcal{R}; m \neq r} a_{lm} Y_{lk} f_m$$

By applying Lemma 1 with x and z as message sources we have:

$$q_{kx}^z \leq min((a_{jr} + b_{jr}) Y_{jk} f_r, \sum_{m \in \mathcal{R}; m \neq r} a_{lm} Y_{lk} f_m) \tag{24}$$

Summing over all the direct client tasks $l \in \mathcal{D}(k)$ ($l \neq j$) yields:

$$b_{kr}^j = b_{jr} Y_{jk} + \sum_{l \in \mathcal{D}(k); l \neq j} q_{kx}^z / f_r \tag{25}$$

By using equations 23, 24, and 25 we obtain:

$$b_{kr} = \sum_{j \in \mathcal{D}(k)} (b_{jr} Y_{jk} + \sum_{l \in \mathcal{D}(k); l \neq j} min((a_{jr} + b_{jr}) Y_{jk}, \frac{\sum_{m \in \mathcal{R}; m \neq r} a_{lm} Y_{lk} f_m}{f_r})) \tag{26}$$

The expression for the lower bound on the throughput of a reference task for a general SRVN can be obtained from equations 21 and 26:

$$f_r^- = \frac{1}{C_r^+} = \frac{1}{S_r + \sum_{k \in \mathcal{N}} S_k [a_{kr} + \sum_{j \in \mathcal{D}(k)} (b_{jr} Y_{jk} + \sum_{l \in \mathcal{D}(k); l \neq j} min((a_{jr} + b_{jr}) Y_{jk}, \frac{\sum_{m \in \mathcal{R}; m \neq r} a_{lm} Y_{lk} f_m}{f_r}))]} \tag{27}$$

The tasks in the SRVN are labelled in such a way that a task j can send messages to a task k only if $k > j$ (see Section 2). For the computation of b_{kr} the non-reference tasks are processed in increasing order of labels; this ensures all the direct client tasks $j \in \mathcal{D}(k)$ are processed prior to the computation of b_{kr}. The computation of b_{kr} starts with the application of equation 26 to the non-reference task k with the smallest label. For a direct client task $j \in \mathcal{R}$, b_{jj} and a_{jj} are defined to be 0 and 1 respectively. Once the processing of task k is complete equation 26 is applied to task $k+1$. The iteration continues and equation 26 is applied to the non-reference tasks in increasing order of labels until all the non-reference tasks in the SRVN are processed.

The computation of the queueing bound for an example SRVN shown in Figure 5 is presented. The SRVN consists of three reference tasks r, m, and n and four non-reference tasks with labels $j < l < v < k$. For the computation of the queueing bound, b_{jr} is computed first and is followed by the computation of b_{lr}, b_{vr}, and b_{kr}. Successive applications of equation 26 to tasks j, l, v, and k yield:

$$b_{jr} = min(Y_{rj}, \frac{Y_{mj} f_m}{f_r})$$

$$b_{lr} = min(Y_{rl}, \frac{Y_{nl} f_n}{f_r})$$

$$b_{vr} = 0$$

$$\begin{aligned} b_{kr} \quad = \quad & [min(Y_{rk}, \frac{Y_{mj} Y_{jk} f_m}{f_r}) + min(Y_{rk}, \frac{Y_{nl} Y_{lk} f_n}{f_r}) + min(Y_{rk}, \frac{(Y_{mv} f_m + Y_{nv} f_n) Y_{vk}}{f_r})] \\ & + [b_{jr} Y_{jk} + min((Y_{rj} + b_{jr}) Y_{jk}, \frac{Y_{nl} Y_{lk} f_n}{f_r}) + min((Y_{rj} + b_{jr}) Y_{jk}, \frac{(Y_{mv} f_m + Y_{nv} f_n) Y_{vk}}{f_r})] \\ & + [b_{lr} Y_{lk} + min((Y_{rl} + b_{lr}) Y_{lk}, \frac{Y_{mj} Y_{jk} f_m}{f_r}) + min((Y_{rl} + b_{lr}) Y_{lk}, \frac{(Y_{mv} f_m + Y_{nv} f_n) Y_{vk}}{f_r})] \end{aligned}$$

The first term within square brackets on the RHS of the previous equation is b^r_{kr}. The second and third terms are b^j_{kr} and b^l_{kr} respectively; since reference task r does not call task v, $b^v_{kr} = 0$. For clarity of presentation we have used b_{jr} and b_{lr} on the RHS of the previous equation; they can be replaced with the appropriate min terms that were derived prior to the derivation of this equation.

Note that for any SRVN the b_{kr} values are functions of reference task throughputs which are unknown. This makes it difficult to compute numerical values for the bounds which is discussed in detail in the following section.

4 Computation of the Bounds

In order to compute the upper bounds on the throughput of reference tasks the a_{jr} coefficients are determined first (see Section 2). Using these computed values of a_{jr} two sets of inequalities are obtained. The first set corresponds to the no contention bounds and the inequalities in this set are obtained by using inequality 4 with every reference task $r \in \mathcal{R}$. The second set corresponds to the utilization-based bounds and each inequality in this set is obtained by using inequality 7 with every non-reference task $j \in \mathcal{N}$. Computation of the lower bounds on reference task throughputs is more complex and is the major focus of attention for this section.

Single Reference Task

For $R = 1$ there is a special case, and a single set of values for f_1^+ and f_1^- can be computed. The no contention and the utilization-based bounds (given by inequalities 4 and 7) imply:

$$f_1^+ = min \left(\frac{1}{S_1 + \sum_{k \in \mathcal{N}} a_{kr1}\ s_{k1}}, \left[\frac{1}{a_{jr}(S_j + \sum_{k \in \mathcal{N}}\ a_{kj1}\ s_{k1})}\ , \text{ for all } j \in \mathcal{N} \right] \right)$$

The term within square brackets on the right hand side of the previous equation denotes a set of values, one for each non-reference task in the SRVN. Since there is only one reference task in the SRVN, equation 15 can be used to yield:

$$f_1^- = \frac{1}{S_1 + \sum_{k \in \mathcal{N}} a_{k1}\ S_k}$$

General cases for $R = 2$

For $R = 2$ it is possible to display the inequalities in an (f_1, f_2) plane as shown in Figure 6(b) for an example SRVN given in Figure 6(a). The no contention-based upper bound on throughput gives rise to the following inequalities:

$$(a)\ f_1 \le \frac{1}{S_1 + Y_{13}\ s_{31}} \quad \text{or} \quad f_1 S_1 + f_1 Y_{13}\ s_{31} \le 1.0$$

$$(b)\ f_2 \le \frac{1}{S_2 + Y_{23}\ s_{31}} \quad \text{or} \quad f_2 S_2 + f_2 Y_{23}\ s_{31} \le 1.0$$

By using the utilization-based upper bound on throughput (inequality 7) and noting that $a_{jr} = Y_{rj}$ yields:

$$(c)\ (Y_{13} f_1 + Y_{23} f_2) \le \frac{1}{S_3} \quad \text{or} \quad (f_1 Y_{13} + f_2 Y_{23}) S_3 \le 1.0$$

The lower bounds on the throughputs are obtained by using equation 27:

$$f_1 \geq \frac{1}{S_1 + S_3 Y_{13}(1 + min[1, \frac{f_2 Y_{23}}{f_1 Y_{13}}])} \tag{28}$$

$$f_2 \geq \frac{1}{S_2 + S_3 Y_{23}(1 + min[1, \frac{f_1 Y_{13}}{f_2 Y_{23}}])} \tag{29}$$

However, the min term in inequality 28 can be expressed as two inequalities (d) and (e):

$$(d)\ f_1 \geq \frac{1}{S_1 + 2S_3 Y_{13}} \quad \text{or} \quad f_1(S_1 + 2S_3 Y_{13}) \geq 1.0$$

$$(e)\ f_1 \geq \frac{1}{S_1 + S_3 Y_{13}(1 + \frac{f_2 Y_{23}}{f_1 Y_{13}})} \quad or \quad f_1 S_1 + f_1 S_3 Y_{13} + f_2 S_3 Y_{23} \geq 1.0$$

Similarly, inequality 29 can give rise to two inequalities:

$$(f)\ f_2(S_2 + 2S_3 Y_{23}) \geq 1.0$$

$$(g)\ f_2 S_2 + f_2 S_3 Y_{23} + f_1 S_3 Y_{13} \geq 1.0$$

All the six inequalities $(a)\ldots(g)$ are linear in f_1 and f_2 and are displayed in the (f_1, f_2) plane in Figure 6(b). The quadrant bounded by the X-axis and the Y-axis is divided by line 1 into two regions. The bounds lie in the upper quadrant if $f_2 Y_{23} > f_1 Y_{13}$, and in the lower quadrant otherwise. The linear constraints, $(a)\ldots(g)$ imposed by the bounds appear as straight lines $a \ldots g$ in the graph. The dark area in Figure 6(b) which is enclosed by the straight lines is termed the *feasible throughput region* for the SRVN. Any feasible set of throughputs, f_1 and f_2, must lie within the feasible throughput region. Figure 6(c) corresponds to the same software architecture shown in Figure 6(a), with a different set of parameters:

$$S_1 = S_2 = 1.0,\ Y_{13} = Y_{23} = 1.0, s_{31} = 1.0, s_{32} = 0$$

As shown in the Figure 6(c) a larger area is enclosed by the throughput region in comparison to Figure 6(b). The size of the feasible throughput region is an indication of the tightness of the bounds which is discussed in more detail in the following section.

For SRVNs with multiple non-reference tasks multiple min terms may appear in the expression for f_r^- (see equation 27). Moreover, the b_{jr} term in the expression for b_{kr} may give rise to nested min terms of the form $min(min(a, b), c)$ in the expression for f_r^-. The nested min term can however be reduced to a simpler form:

$$min(min(a, b), c) = min(a, b, c)$$

If the argument a, b, or c involves another min operation the process of reduction is continued for one more round. For example, if $a = min(d, e)$ then $min(a, b, c)$ is expressed as $min(d, e, b, c)$. For each min term in the expression for f_r^- the reduction process is continued until none of the arguments of the min operation includes another min operator. Then we can proceed as in the case of the simple SRVN in Figure 6(a) and replace the expression for each f_r^- with a number of inequalities. Each inequality would represent a particular choice as to which argument is the minimum for each min operation in the expression for f_r^-.

Large Values of R: Linear Inequalities

For SRVNs with more than two reference tasks a graphical presentation of the feasible throughput region becomes impossible. However, as described above for $R = 2$, the upper and lower bounds can be decomposed into a number of inequalities which are linear in the throughputs $f_1 \ldots f_R$. Although this set of inequalities is a less compact expression for the bounds than that captured in inequalities 4, 7 and equation 27, it demonstrates the convexity of the feasible throughput region.

The set of inequalities corresponding to the upper bound and the lower bound on throughput could be used in a phase-1 iteration of a linear programming algorithm to find a single feasible point. However, it is not straightforward to extract the linear inequalities, and the linear programming phase-1 solution only gives a single point, and not the entire feasible region. Thus, this approach was dropped in favour of the box bounds which are described next. Another advantage of the box bounds is that they are separable. Each reference task throughput has its own limits, without interactions, which makes the results more understandable and usable.

Interval Arithmetic and Box Bounds

As an alternative to the linear programming based solution an automated solution may be found for a relaxed set of bounds (larger than the true bounds) which we term *box bounds.* The box bounds are based on a set of box constraints which is a rectangular polytope in an R-dimensional space, circumscribed around the true feasible throughput region. Based on these constraints a single pair of bound values, F_r^+ and F_r^- can be found for each reference task r:

$$F_r^+ \geq f_r \geq F_r^-$$

F_r^+ and F_r^- are called the box bounds and correspond to the lowest and the highest feasible values of a single f_r. The box constraints for the example SRVN shown in Figure 6(a) is a rectangle as shown in Figure 7. The box bounds can be obtained from the coordinates of the two end points of the diagonal (F_1^+, F_2^- and F_1^-, F_2^+) for this rectangle.

We experimented with an automated method based on interval arithmetic computations for the determination of the box bounds. The BNR Prolog language [1] has an interval arithmetic capability that is centred around a data type called *interval* which is an object representing a real number lying between an upper and a lower bound. The reference task throughputs are represented by intervals in the program and each linear inequality corresponding to an upper bound (of the form of 4 or 7) or the lower bound (based on 27) is represented by a Prolog subgoal. The system interprets all these subgoals and produces a set of values $F_r^+ \geq f_r \geq F_r^-$ (for every $r \in \mathcal{R}$) such that F_r^+ and F_r^- are the highest and the lowest values for f_r that satisfy all the inequalities. As each inequality is evaluated computation proceeds forward and intervals can change values by narrowing. An interval is narrowed when its lower bound is raised or its upper bound is lowered or both. BNR Prolog uses backtracking which may undo the narrowing of an interval as additional constraints are evaluated.

The box bounds for the SRVN of Figure 6(a) (Case I) and a more complex SRVN shown in Figure 8 (Case II) are presented in Table 1. The midpoint of the interval bounded by the box bounds is potentially useful as an estimate for the reference task throughputs. For the sake of comparison Petri Net models of the two task systems were constructed and expected task throughputs were computed analytically by the GSPN package [6]. The reference task throughputs computed from the Petri net model are included in Table 1. For the four throughput values presented in Table 1, the largest deviation between the throughput estimates based on the box bounds and the exact task throughputs computed by the GSPN package was observed to be 13%.

Table 1: Examples of Box Bounds

Case	F_1^-	F_1^+	F_2^-	F_2^+	midpoint $[f_1,\ f_2]$	GSPN $[f_1,\ f_2]$
I (Fig.6(a))	0.034013	0.038889	0.055555	0.071429	[0.036451, 0.063492]	[0.0374332, 0.0561497]
II (Fig.8)	0.00018945	0.00034262	0.0016059	0.0026075	[0.000266, 0.0021067]	[0.0002854, 0.001929]

Both the method based on linear programming as well as the box bounds are applicable to any general arbitrary SRVN. The computation of the bounds is somewhat simpler in special cases for which the reference task throughputs are related to one another. Investigation of two such special cases of SRVNs is presented next.

Proportional Reference Task Throughputs

For $R = 2$ and higher, a special tractable case arises if the reference task throughputs have constant ratios. This may come about in several ways. A certain set of throughput ratios may be desired for performance planning reasons. Values for ratios might be assumed temporarily, as an aid to exploring the feasibility space, or for producing a cross-section of it for graphical display. If the ratios $\gamma_r = f_r/f_1$ are known constants for $r = 2, \ldots R$, a single-variable problem is obtained which is almost as simple as the case in which $R = 1$. Returning to equation 27 we find that each "min" term in the denominator of the RHS of the equation is now of the form:

$$min((a_{jr} + b_{jr})Y_{jk}, \frac{\sum_{m\in\mathcal{R};m\neq r} a_{lm}Y_{lk}f_m}{f_r})) = min((a_{jr} + b_{jr})Y_{jk}, \frac{\sum_{m\in\mathcal{R};m\neq r} a_{lm}Y_{lk}\gamma_m}{\gamma_r}))$$

Since all the quantities are known, the min expressions can be evaluated and b_{jr} can be computed from inequality 26 for tasks in increasing order of task labels. Thus f_1^- can be found. The upper bounds can similarly be resolved to give f_1^+. Then bounds on the throughputs of other reference tasks are given by

$$f_1^+\gamma_r \geq f_r \geq f_1^-\gamma_r$$

To sweep out a cross-section of the feasible throughput region for graphical visualization, the vector of values γ_r can be "swept" through a sequence of directions in the throughput space. The bounds are found for each direction, and the ensemble of results is plotted as a set of line segments. Each segment represents the feasible values along one direction. This technique is easily generalized to line-segments in any direction in the throughput space, to sweep out a cross-section of arbitrary orientation.

Symmetrical Cases

For large R, symmetrical cases can be analyzed and displayed by the same methods as for $R = 1$. In such a system the reference tasks are statistically identical in the sense that they are characterized by the same mean computation times and the number of messages generated in a cycle. By symmetry, we set $\gamma_2 = \ldots = \gamma_R = 1$ for the relative throughputs, and we need examine only the inequalities for f_1, using the approach of the previous paragraph.

More complex symmetrical cases can also be reduced using the same approach. If the parameters are symmetrical with respect to a certain group of reference tasks $\mathcal{R}' = \{r, r+1, \ldots r'\}$ then these statistically identical tasks can be grouped together to form a class as in multiclass queueing networks. The delays and throughputs of the tasks in each group will be equal, and one can substitute f_r for $f_{r+1}, \ldots f_{r'}$, and proceed with fewer variables. If there are just two symmetrical groups, the bounds for each group can be plotted as in Figure 6(b).

5 Tightness of the Bounds

The bounding technique presented in the previous section is an effective tool for the performance evaluation of SRVN systems. The tightness of the bounds depends however, on the nature of the system under investigation. Two case studies each of which deals with a generic software architecture are presented in this section. The ratio between the upper and lower bounds on the throughput of a given reference task, r, represented by θ_r, is an indicator of the tightness of the bounds for r. The average of these ratios for all the reference tasks, $\theta = \sum_{r \in \mathcal{R}} \theta_r / R$, represents a measure of tightness of bounds for the entire SRVN.

5.1 The Client-Server Architecture

The *client-server* architecture is characterized by two classes of tasks. The reference tasks $1, .., R$ are called client tasks which employ the services of a single server task N (see Figure 9). Such a software architecture may represent the interaction between a number of application tasks and a server task that provides an interface for a shared hardware device. The tightness of the bounds for this software architecture is presented. By using inequality 21 we have:

$$f_r^- = \frac{1}{S_r + S_N Y_{rN} + S_N b_{Nr}} \tag{30}$$

Using equation 26 and noting that $a_{Nr} = Y_{rN}$ and $a_{Nl} = Y_{ln}$ $(for\ l \in \mathcal{R})$ we get:

$$b_{Nr} = \sum_{l \in \mathcal{R}\ ;\ l \neq r} min(Y_{rN}, \frac{Y_{lN} f_l}{f_r}) \tag{31}$$

By using the no contention and the utilization-based bounds the upper bound on f_r can be expressed as:

$$f_r^+ = \frac{1}{S_r + s_{N1} Y_{rN} + W_{rN}^-} \tag{32}$$

where W_{rN}^- is the lower bound on the queueing delay experienced per cycle by task r at task N. Equations 30 and 32 yield:

$$\theta_r = 1 + \frac{Y_{rN}(S_N - s_{N1}) + b_{Nr} S_N - W_{rN}^-}{S_r + Y_{rN}\ s_{N1} + W_{rN}^-}$$

By using equation 31 and noting that $W_{rN}^- \geq 0$:

$$\theta_r \leq 1 + \frac{Y_{rN}(R S_N - s_{N1})}{S_r + Y_{rN}\ s_{N1} + W_{rN}^-} \tag{33}$$

A small θ_r signifies tight bounds on reference task throughputs whereas a larger value of θ_r is obtained when the bounds are relatively loose. Substituting 32 in 33 and by summing over all reference tasks yields:

$$\sum_{r=1}^{R} \theta_r \leq R + \sum_{r=1}^{R} (R\, S_N - s_{N1}) Y_{rN} f_r^+$$

By substituting U_N^+ for $\sum_{r=1}^{R} S_N Y_{rN} f_r^+$ and u_{N1}^+ for $\sum_{r=1}^{R} s_{N1} Y_{rN} f_r^+$ we get:

$$\sum_{r=1}^{R} \theta_r \leq R + R\, U_N^+ - u_{N1}^+$$

From the previous equation it follows:

$$\theta \leq 1 + U_N^+ - \frac{u_{N1}^+}{R} \tag{34}$$

U_N^+ is the maximum useful utilization of task N whereas u_{N1}^+ is the maximum useful utilization of task N in its first phase of execution. Note that the useful utilization of a task is the proportion of time the task is in execution or is blocked awaiting the reply of a message from another task. Inequality 34 implies that the bounds on reference task throughputs are tight when U_N^+ is small. Since $U_N^+ \leq 1.0$ and $u_{N1}^+ \geq 0$, the average ratio between the upper and the lower bounds, θ, can be at most a factor of 2. The highest value of θ is achieved for a homogeneous system in which the reference tasks are statistically identical in the sense that they are characterized by the same mean and distribution for the computation times and message rates. For such a system the expected throughputs of the reference tasks are equal which achieves an upper bound on b_{Nr} (see equation 31).

It can be verified that $\theta = \theta_r$ (for $\forall r \in \mathcal{R}$) $= 2$ is obtained for a homogeneous system in which:

$$S_r = R Y_{rN} S_N \text{ and } s_{N1} = 0$$

Lower values of this ratio can be expected for systems in which the first phase of the server task is dominant ($U_N^+ \approx u_{N1}^+$) and the number of client or reference tasks in the system, R, is small. For example, if there are only two client tasks and the server task has only a first phase of execution, $\theta_r \leq 1.5$.

Numerical results for a homogeneous client-server system consisting of four client tasks are presented in Table 2. Three different cases are considered. In each of these cases $Y_{rN} = 1.0$. The total work done per cycle of a reference task, given by, $S_r + Y_{rN} S_N$, is maintained to be a constant in all these cases. For Case I the server task has a non-zero computation time for each of its two phases of execution, whereas for Case II the computation time for the server task in the second phase is zero and for Case III the computation time for the server task in the first phase is zero. For all the three cases as, U_N^+, the maximum useful utilization of the server task increases, θ is observed to increase. For the same set of parameter values largest values of θ are produced for Case III in which the first-phase computation time of the server task is 0. After U_N^+ reaches 1.0, further increase in S_N increases the contention at the server task producing lower values of θ in all the three cases.

Table 2: Tightness of Bounds for the Homogeneous Client-Server Architecture

Case	S_r	s_{N1}	s_{N2}	U_N^+	f_r^-	f_r^+	θ
I	10.0	0.1	0.1	0.0798	0.092592	0.09901	1.0693
	8.0	1.1	1.1	0.9671	0.059523	0.10990	1.8464
	6.0	2.1	2.1	1.0000	0.043859	0.059524	1.3572
	4.0	3.1	3.1	1.0000	0.034722	0.040323	1.1613
	2.0	4.1	4.1	1.0000	0.028735	0.030488	1.0610
	0.2	5.0	5.0	1.0000	0.024875	0.025001	1.0051
II	10.0	0.2	0	0.0784	0.092592	0.09804	1.0588
	8.0	2.2	0	0.8628	0.059523	0.09804	1.6471
	6.0	4.2	0	1.0000	0.043859	0.059524	1.3572
	4.0	6.2	0	1.0000	0.034722	0.040323	1.1613
	2.0	8.2	0	1.0000	0.028735	0.030488	1.0610
	0.2	10.0	0.0	1.0000	0.024875	0.025001	1.0051
III	10.0	0	0.2	0.0800	0.092592	0.10001	1.0801
	8.0	0	2.2	0.9979	0.059523	0.11364	1.9092
	6.0	0	4.2	1.0000	0.043859	0.059524	1.3572
	4.0	0	6.2	1.0000	0.034722	0.040323	1.1613
	2.0	0	8.2	1.0000	0.028735	0.030488	1.0610
	0.2	0	10.0	1.0000	0.024875	0.025001	1.0051

5.2 The Hierarchical Architecture

An extension of the client-server architecture called the *hierarchical architecture* is considered here. This software architecture is characterized by three levels of tasks (see Figure 10). At the top level are R reference tasks and there is a set N of pure server tasks at the lowest level. A reference task can communicate with a pure server task only through task a which forms the intermediate level in the hierarchy. Such an SRVN can be used to model a system in which a number of application tasks uses the services of a number of different server tasks. The service requests from an application task are received by a "manager" task which controls the access to the different server tasks. The tasks in a given level are statistically identical and are characterized by the same s_{ip} and y_{ijp} values. The reference tasks $r \in \mathcal{R}$ are characterized by their mean computation times S_r and and the mean number of messages, Y_a, that are sent to task a in a single cycle. Task a on an average generates Y_n messages per cycle for each pure server task $k \in \mathcal{S}$. y_{n1} messages out of Y_n are generated in the first phase of execution for task a. The mean execution times of each pure server task are characterized by s_{n1} for phase-1 and S_n for the entire cycle. Using an analysis similar to that used in the case of the client-server architecture it can be verified that the upper and lower bounds on the throughput of a reference task are given by the following equations.

$$f_r^+ = \frac{1}{S_r + (s_{a1} Y_a + W_{ra}^-) + N\ Y_a\ y_{n1}\ s_{n1}}$$

$$f_r^- = \frac{1}{S_r + (S_a Y_a + S_a b_{ar}) + N S_n (Y_a Y_n + b_{kr})}$$

where,

$$b_{ar} = (R-1)Y_a \;\text{ and }\; b_{kr} = (R-1)Y_aY_n \; for\; k \in \mathcal{S}$$

In a homogenous system the reference tasks are statistically identical and $\theta = \theta_r$ (for $\forall \in \mathcal{R}$). From the previous equations, it can be easily verified that the ratio between the upper and lower bounds on the throughput of a reference task satisfies the following relationship:

$$\theta = \theta_r \leq 1 + U_a^+ - \frac{u_{a1}^+}{R} + N(U_k^+ - u_{k1}^+) \tag{35}$$

$$\text{where } U_a^+ = f_r^+ R Y_a (S_a + N Y_n\, s_{n1}),\; U_k^+ = f_r^+ R Y_a Y_n S_n,$$

$$u_{a1}^+ = R f_r^+ Y_a (s_{a1} + N\, y_{n1}\, s_{n1}),\; u_{k1}^+ = f_r^+ R Y_a\, Y_n\, s_{n1}$$

U_a^+ and U_k^+ represent the maximum useful utilization of the manager task a and each pure server task respectively. Similarly, u_{a1}^+ and u_{k1}^+ represent the maximum useful utilizations of the manager task and a server task in their first phases of execution only. Since U_a^+ as well as U_k^+ are less than or equal to 1.0, it follows from inequality 35 that the ratio between the upper and lower bounds on reference task throughputs cannot exceed $2 + N$. As in the case of the client-server architecture, the tightness of the bounds is improved for an SRVN if the number of reference tasks is small and the first phases of non-reference tasks are dominant ($U_a \approx u_{a1}$ and $U_k \approx u_{k1}$).

Numerical results for a system with a hierarchical architecture are presented in Table 3. The number of reference tasks, R, is 4, and the number of pure server tasks N, is 2. Five different cases are considered. For all these cases, $Y_a = Y_n = y_{n1} = 1.0$. The total work done per cycle of a reference task is kept the same for each entry recorded in Table 3. Case I and Case II correspond to situations in which $S_N = 0$ whereas Case III and Case IV correspond to the situations in which $S_a = 0$. The effects of an increase in the computation time in the first phases of execution of the non-reference tasks are presented in Case I and Case III, while the effects of an increase in the computation time in the second phases of execution are presented in Case II and Case IV. Case V considers an intermediate situation in which each phase of execution of a given non-reference task is characterized by the same computation time: $s_{a1} = s_{a2}$ and $s_{n1} = s_{n2}$.

As in the case of the client-server architecture, θ is small when the maximum useful utilizations of the non-reference tasks are small. Inequality 35 indicates that large values of θ may be obtained when the non-reference tasks are heavily utilized and are dominated by their second phases of execution. The largest observed value of $\theta = 2.714$ is obtained in Case IV for which $S_N = s_{n2}$; this is in agreement with inequality 35 which (with $U_a^+ = 0$) implies $\theta \leq 3$. In all the five cases θ increases with an increase in the utilizations of non-reference tasks and then starts decreasing as more and more queuing occurs at the non-reference tasks. A comparison of the same rows of Case I and Case II shows that, a higher value of θ may be obtained for the case in which the post-rendezvous phase is dominant ($s_{a1} = 0$) especially when U_a is small. A similar observation can be made in the context of the pure server tasks from Case III and Case IV.

5.3 Comparison with Queueing Network Model

The dual nature (customer as well as server) of tasks in an SRVN and the existence of the post-rendezvous phase make it difficult to model their behavior by standard queueing network models. The client-server architecture, however, can be modelled approximately by a closed queueing network. The client or the reference tasks in the system behave purely as customers with think times of S_r and the server task is modelled as a queueing center. The queueing network model is approximate in the sense that no distinction is made between the two phases of execution of the server task and the service demand for the queueing center is set to S_N. A comparison of the bounds for an SRVN model of a homogeneous client-server system with the bounds obtained from a single class closed queueing network model of the system is presented in Figure 11. Since minimal assumptions are made about the distributions of task execution times and message rates in the SRVN, the asymptotic bounding analysis for single class queueing networks [23] is used. The asymptotic bounds on the throughput of the system are computed from the queueing network model as [11]:

$$\frac{R}{R\,Y_{rN}S_N + S_r} \leq X(R) \leq min(\frac{1}{Y_{rN}\;S_N}, \frac{R}{Y_{rN}S_N + S_r})$$

The system throughput $X(R)$ is defined to be the total number of cycles completed per unit time by the R reference tasks in the system. By using inequalities 4, 7, and equation 27 the SRVN bounds are computed as:

$$\frac{R}{R\,Y_{rN}S_N + S_r} \leq X(R) \leq min(\frac{1}{Y_{rN}\;S_N}, \frac{R}{Y_{rN}\;s_{N1} + S_r})$$

The upper bound on system throughput $X(R)^{+}$ and the lower bound on system throughput $X(R)^{-}$ computed from the SRVN model as well as the bounds on system throughput obtained from the queueing network model are presented in Figure 11. S_r is chosen to be 5.0, the mean number of rendezvous calls made by a reference task per cycle, Y_{rN}, is fixed at 1.0 and the total computation time (S_N) of the server task is held at 1.0. The chosen set of parameters is appropriate for the comparison between the two models. Similar results can be obtained with other parameter values as well. Three different cases are considered for the SRVN model. Each of these cases uses a different set of values for s_{N1} and s_{N2}. As expected, the lower bound on $X(R)$ computed from the SRVN model is exactly the same as the lower bound obtained from the queueing network model. The upper bound on system throughput, $X(R)$, computed from the SRVN model for the case in which $s_{N1} = 1.0, s_{N2} = 0$ is the same as the upper bound on $X(R)$ obtained from the queueing network model. The upper bounds computed from the two models differ, however, when s_{N2} is non-zero and the number of reference tasks is small. Note that when the computation time in the second phase for the server task is non-zero the approximate queueing network model of the multitasking system under estimates the system throughput and the upper bound computed from this model may not be the true bound. The difference between the upper bounds on system throughput computed by the two models decreases however as R increases. For a small number of reference tasks the upper bound on throughput computed from the SRVN model increases as more and more work is transferred from the first phase of the server task to its second phase. The upper bound seems to be less sensitive to the distribution of work between the two phases of the server tasks for large values of R. As the number of reference tasks in the system increases more and more queueing occurs at the server task and the upper bound on $X(R)$ obtained (from the SRVN model) for different distributions of work between the two phases of the server task approach the upper bound computed from the queueing network model.

6 Processor Contention

The performance analysis of SRVNs presented in the previous sections has concentrated only on the software resources in the system. It was assumed so far that an unlimited number of processors is available such that every task in the system is allocated on a separate processor. The effect of a limited number of processors and processor contention is considered in this section. Tasks are assumed to be allocated statically to processors. If the processor is busy when a task wants to execute it joins a FIFO queue of ready to run tasks that are allocated on the same processor. When the task running on a processor blocks for a rendezvous the processor is allocated to the task at the head of the ready to run queue. In order to account for processor contention the SRVN graph is modified to include a special class of nodes called *hardware servers*. Each processor is represented by a hardware server node which is shown as a rectangle in the SRVN graph (see Figure 12). Without loss of generality this set $\mathcal{H}$ of hardware server nodes are labelled from $N+1$ to $N+H$ in the graph. A hardware server behaves like a pure server task in the sense that it does not receive service from any other task and is special because it exhibits different service times for its different "customers". A task j behaves as a customer for the hardware server h that models the processor on which it is allocated and the interaction between the task and its processor is represented by an *execution edge* (shown by a bold arrow from j to h in the modified SRVN graph).

Computation of the Upper Bounds

The no-contention bound defined in Section 3.1 ignores queueing of tasks for resources and is thus independent of how tasks are allocated to processors. Inequality 4 can still be used to compute the no-contention bound for each reference task $r \in \mathcal{R}$.

The utilization-based bound discussed in Section 3.1 considers the software resources in the system only and gives rise to an inequality of the form given by inequality 7 for every non-reference task in the system. When processor contention is considered the utilization based bound gives rise to a set of additional inequalities, one for every processor in the system. If $\mathcal{D}(h)$ denotes the set of tasks that are allocated on the processor modelled by the hardware server h then:

$$\sum_{j \in \mathcal{D}(h)} S_j \, f_j \leq 1.0$$

S_j is the total computation time for task j in a single cycle. By using the expression for a_{jr} from equation 1 and defining $a_{rr} = 1$, we obtain:

$$\sum_{j \in \mathcal{D}(h)} (S_j \sum_{r \in \mathcal{R}} a_{jr} f_r) \leq 1.0 \;\text{ for all }\; h \in \mathcal{H}$$

Computation of The Lower Bound

Computation of the lower bound on reference task throughputs is more complicated and requires the computation of an upper bound on the queueing delay experienced by tasks. In order to obtain this bound it is required to characterize task behavior at a finer level, in terms of processor requests generated by tasks. A task's request for execution on its processor is modelled as a rendezvous between two tasks. Task j, for example, is considered to initiate a rendezvous with its processor h whenever it is ready to execute. There is only a phase-1 service provided for such a rendezvous by the hardware server. The service time for this rendezvous between task j and hardware server h is denoted by σ_j, the expected service time per visit of the processor. The expected number of such rendezvous per cycle of j is denoted by Y_j which is equal to the expected number of processor requests generated by j in a single cycle. The execution edge between j and h in the modified

SRVN graph is labelled with σ_j and Y_j. Since a task $j \in \mathcal{R} \cup \mathcal{N}$ executes on its processor only before and after it has performed a rendezvous with a task $k \in \mathcal{N}_j$

$$Y_j = \sum_p [(\sum_{k \in \mathcal{N}_j} y_{jkp}) + 1]$$

For the sake of simplicity it is assumed that the processor allocated to a task is rescheduled after the end of every phase of execution. A reference task r has only a single phase of execution and the expected service time per visit at its processor is given by:

$$\sigma_r = \frac{S_r}{Y_r} \text{ for all } r \in \mathcal{R}$$

A non-reference task j has two phases of execution and the expected service time per visit at the processor, σ_{jp}, can be different for each phase of execution:

$$\sigma_{jp} = \frac{s_{jp}}{\sum_{k \in \mathcal{N}_j} y_{jkp} + 1} \text{ for all } j \in \mathcal{N}$$

For the computation of the lower bound on reference task throughputs the service time per visit at the processor is defined to be σ_j for both phases of execution of task j:

$$\sigma_j = max(\sigma_{j1}, \sigma_{j2})$$

After modifying the SRVN graph for modelling the interaction between a task and its processor we follow the developments presented in Section 3.2 for the computation of the lower bound on reference task throughputs. First it is noted that for systems with a single reference task the upper bound on the cycle time of the reference task corresponds to a sequential execution of the task system modelled by the SRVN. Thus irrespective of the way tasks are coallocated to processors:

$$C_r^+ = S_r + \sum_{k \in \mathcal{N}} a_{kr} S_k$$

For a system with more than one reference task, $(\sum_{j \in \mathcal{N}_r} W_{rj})^+$, the interference produced by other reference tasks in the system needs to be computed:

$$C_r^+ = S_r + \sum_{k \in \mathcal{N}} a_{kr} S_k + (\sum_{j \in \mathcal{N}_r} W_{rj})^+ \tag{36}$$

A task can queue up for a processor when it wants to perform its own computation and for another task when it generates a rendezvous request. The recipient of this message in turn generates a request for its processor to service the rendezvous request. Thus the queues at tasks are transferred to the queues at their respective processors which execute and contribute to $(\sum_{j \in \mathcal{N}_r} W_{rj})^+$. Note that in the context of processor contention, $\mathcal{N}_r$ in the expression for $(\sum_{j \in \mathcal{N}_r} W_{rj})^+$ includes task r and W_{rr} corresponds to the contention experienced by r when it visits the processor for its own computation.

In order to compute the contribution of each hardware server to $(\sum_{j \in \mathcal{N}_r} W_{rj})^+$, b_{kr} is computed for every $k \in \mathcal{N}$ in exactly the same manner as described in Section 3.2. Once b_{kr} for all the tasks that are allocated on the processor modelled by the hardware server h are known we could follow the procedure outlined in Section 3.2 and apply equation 25 to compute b_{hr}^k, the upper bound on the number of contending processing requests that delay a cycle of r when r is blocked waiting for k to execute on h. However, the expected time to service each of these processing requests at h is not a constant but is dependent on the task that generates the request. Thus instead of computing b^k_{hr} we compute the upper bound on the delay caused by the other reference tasks when k visits the hardware server h for processing the rendezvous messages initiated by r. Note that this delay includes all the delays experienced by r in one cycle in performing rendezvous with k through all the existing paths between r and k. The upper bound on the delay is given by:

$$b_{kr} Y_k \sigma_k + \sum_{l \in \mathcal{D}(h); l \neq k} min((a_{kr} + b_{kr}) Y_k, \frac{\sum_{m \in \mathcal{R};\, m \neq r} a_{lm} Y_l f_m}{f_r}) \sigma_l$$

[Note that when r visits h directly for its own execution (i.e., for $k = r$), $a_{rr} = 1$, and $b_{rr} = 0$]. The first component of this upper bound, $b_{kr} Y_k \sigma_k$, corresponds to the service (by k) of the contending messages initiated by the other reference tasks that delay the rendezvous between r and k. The second component corresponds to the additional delay introduced at the processor due to the processing requests generated by the other tasks $l \in \mathcal{D}(h)$ ($l \neq k$) that are allocated on the same processor as task k. Note that each task l that contends with the visits of k at h behaves as a separate "customer" with its own execution time per visit σ_l. By repeating the computation for each task $k \in \mathcal{D}(h)$ the contribution of hardware server h to $(\sum_{j \in \mathcal{N}_r} W_{rj})^+$ is obtained as:

$$\sum_{k \in \mathcal{D}(h)} [b_{kr} Y_k \sigma_k + \sum_{l \in \mathcal{D}(h); l \neq k} min((a_{kr} + b_{kr}) Y_k, \frac{\sum_{m \in \mathcal{R};\, m \neq r} a_{lm} Y_l f_m}{f_r}) \sigma_l]$$

Summing over all the hardware servers in the SRVN yields:

$$(\sum_{j \in \mathcal{N}_r} W_{rj})^+ = \sum_{h \in \mathcal{H}} \sum_{k \in \mathcal{D}(h)} [b_{kr} Y_k \sigma_k + \sum_{l \in \mathcal{D}(h); l \neq k} min((a_{kr} + b_{kr}) Y_k, \frac{\sum_{m \in \mathcal{R};\, m \neq r} a_{lm} Y_l f_m}{f_r}) \sigma_l] \quad (37)$$

By using the previous equation in conjunction with equation 36 we obtain the following expression for the lower bound on the throughput of reference task r:

$$f_r^- = \frac{1}{S_r + \sum_{k \in \mathcal{N}} a_{kr} S_k + \sum_{h \in \mathcal{H}} \sum_{k \in \mathcal{D}(h)} [b_{kr} Y_k \sigma_k + \sum_{l \in \mathcal{D}(h); l \neq k} min((a_{kr} + b_{kr}) Y_k, \frac{\sum_{m \in \mathcal{R};\, m \neq r} a_{lm} Y_l f_m}{f_r}) \sigma_l]} \quad (38)$$

The effect of processor contention on the throughput bounds is explored in the context of an example SRVN with a homogeneous client-server architecture (see Figure 12(a)). The parameters for the model are:

$$for\ r = 1,..,4:\ S_r = 8.0\ \ Y_{r5} = 1.0,\ s_{51} = 1.1\ and\ s_{52} = 1.1$$

Figure 12(b) shows the SRVN after the addition of three hardware servers each of which represents a processor. The parameters associated with the execution edges in the modified SRVN are given by:

$$for\ r = 1 \ldots 4:\ \sigma_r = 4\ Y_r = 2.0,\ \sigma_5 = 1.1,\ Y_5 = 2.0$$

Table 4 shows the results for four different cases ($H = 1$, $H = 2$, $H = 3$, and $H = 5$) each of which corresponds to a specific number of processors on which the multi-tasking system executes. Given the total number of processors, tasks can be allocated to the processors in different ways. Since the server task is used by every client task, it is allocated a separate processor when there are more than a single processor in the system. The remaining number of processors are distributed equally among the client tasks which are statistically identical to one another. In the first case ($H = 1$) all the tasks are allocated on a single processor. In the second case ($H = 2$) all the client tasks are allocated on a single processor whereas in the third case ($H = 3$) two client tasks are allocated on a single processor. In the fourth case ($H = 5$) each task executes on its own processor. For each of these cases the expected throughputs of the reference tasks are equal to one another. This reduces the computation of the bounds to a single variable problem and closed form expressions are obtained for f_r^-, f_r^+ as well as $\theta = \theta_r = f_r^+ / f_r^-$(for $\forall r \in \mathcal{R}$). An SRVN with four processors is different from these cases in the sense that reference task throughputs are not available in closed form and the box bounds based on the interval arithmetic technique discussed in Section 4 need to be computed.

The lower bound on reference task throughputs obtained with $H = 1$ is pessimistic in the sense that C_r^+ is larger than the sequential execution time for the multi-tasking system; this leads to a higher value of θ in comparison to the other cases presented in Table 4. From the values of θ computed for the last three cases the tightness of the bounds seems to improve as the number of processors in the system is reduced.

7 Summary and Conclusions

Upper and lower bounds on reference tasks throughputs of rendezvous-based intercommunicating task systems were analyzed. These bounds are robust in the sense that they rest on such weak stochastic assumptions about the behavior of the system and are thus applicable to a large variety of rendezvous-based systems. Techniques for the computation of the bounds when each task is allocated its own processor were presented first. The effect of limited number of processors was incorporated later into the model. The inter-dependence among reference task throughputs makes it difficult to extract the bounds in closed analytic form. Different methods for the computation and presentation of the bounds were discussed. The graphical display of a feasible throughput region was found to be useful for a system with only two classes of reference tasks. The notion of box bounds and an interval arithmetic based technique for the computation of the bounds were presented for handling general arbitrary SRVNs.

The application of the bounds to performance evaluation was performed in the context of the client-server and the hierarchical software architectures. Closed form expressions for the tightness of the bounds were obtained in both cases. The average ratio between the upper and lower bounds on the throughputs of a reference task was used as a measure of tightness of the bounds. For the client-server architecture the average ratio between the upper and lower bounds on reference task throughputs was observed to be at most a factor of 2. This highest value of the ratio is realized when the reference tasks in the system are statistically identical. The bounds can be expected to be tighter when the behaviors of the reference tasks in the system are dissimilar. Analysis of the hierarchical architecture showed that the bounds are potentially loose in comparison to the client-server case. The maximum useful utilizations of the non-reference tasks were observed to have a strong inter-relationship with the tightness of the bounds. A reference task cycle consists of computations performed at the reference task as well as at the non-reference tasks in the system; each software architecture was investigated for different distributions of a fixed amount of work (per cycle) between the reference and the non-reference tasks. For both the client-server and the hierarchical software architectures the bounds were observed to be tight when most of the work in the cycle is performed at the reference task and the maximum useful utilizations of the non-reference tasks are small. The ratio between the upper and lower bounds increases, initially, as more and more work is transferred from the reference tasks to the non-reference tasks. However the ratio was observed to reach a peak value and further increase in computation times of the non-reference tasks decreases the ratio between the upper and lower bounds on reference task throughput. These observations indicate that given a fixed amount of work to be performed per cycle of a reference task, tight bounds can be obtained when the distribution of work is unbalanced: most of the work is performed either at the reference task or at the non-reference tasks. The division of work between the two phases of execution of non-reference tasks has a secondary effect on the tightness of bounds. Tighter bounds are obtained in case of both the software architectures when the in-rendezvous phase is more dominant than the post-rendezvous phase.

In the analyses presented in this paper the non-reference tasks were assumed to be homogeneous in the sense that the behavior of a receiver task is independent of the task which has requested the rendezvous. Extending the bounding analysis to tasks with ports or entries each of which may be characterized by a different set of parameters forms an important direction for further research. The effect of congestion in the communication medium was not modelled explicitly in this paper. Analysis of the effect of congestion due to a shared communication medium warrants further investigation.

Acknowledgements

We would like to thank the referees whose comments led to a significant improvement in the clarity of presentation. Thanks are due to Bell-Northern Research Ltd. for donating the BNR Prolog software to the university and to Gerald Karam for his help in using it. This research was supported by grants from the Telecommunications Research Institute of Ontario through its project on Telecom Software Methods.

References

[1] *BNR Prolog Reference Manual* Computing Research Laboratory, Bell Northern Research Ltd., Ottawa, Canada, 1988.

[2] , *INMOS Limited Occam 2 Reference Manual.* 66 Wood Land End, Hemel Hempstead, Hertfordshire, England HP2 4RG:Prentice Hall International, 1988.

[3] C.M. Woodside, "Throughput Calculation for Basic Stochastic Rendezvous Networks," *Performance Evaluation*, Vol. 9, pp. 143–160, 1989.

[4] C.M. Woodside, J.E. Neilson, D.C. Petriu, and S. Majumdar, *The Rendezvous Network Model for Performance Synchronous Multi-Tasking Distributed Software*, SCE-89-8, March 1989.

[5] C.M. Woodside, J.E. Neilson, J.W. Miernik, D.C. Petriu, and R. Constantin, "Performance of Concurrent Rendezvous Systems with Complex Pipeline Structures," In *Proc. 4th International Conference in Modelling Techniques and Tools for Computer Performance Evaluation*, pp. 361 – 378, September 1988.

[6] G. Chiola, *GreatSPN Users' Manual, Version 1.3.* Dipartimento di Informatica, Università degli Studi di Torino, Torino, Italy, September 1987.

[7] G. Ciardo *Analysis of Large Stochastic Petri Net Models* PhD thesis, Department of Computer Science, Duke University, 1989.

[8] DOD, *Reference Manual for the Ada Programming Language ANSI/MIL-STD 1815.* United States Department of Defense, January 1983.

[9] D.R. Cheriton, *The Toth Operating System: multi-process structuring and portability.* New York:Elsevier, 1982.

[10] D.R. Cheriton, "The V kernel: A software base for distributed systems," *IEEE Software*, Vol. 1, No. 2, pp. 19–42, April 1984.

[11] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.G. Sevcik, *Quantitative System Performance.* Englewood Cliffs, N.J. 07632:Prentice-Hall, Inc., 1984.

[12] P. Heidelberger and K.S. Trivedi, "Analytic Queueing Models for Programs with Internal Concurrency," *IEEE Transactions on Computers*, Vol. C-32, No. 1, January 1983.

[13] J.A. Rolia, "Performance Estimates For Systems With Software Servers: The Lazy Boss Method," *Proceedings VIII SCCC International Conference on Computer Science*, July 1988.

[14] J.W. Miernik, C.M. Woodside, J.E. Neilson, and D.C. Petriu, *Performance of Stochastic Rendezvous Networks with Priority Tasks*, Report, Ottawa, Canada K1S 5B6, May 1988.

[15] J.W. Miernik, C.M. Woodside, J.E. Neilson, and D.C. Petriu, "Throughput of Stochastic Rendezvous Networks with Caller-Specific Service and Processor Contention," In *Proc. IEEE Infocom '88*, pp. 1040–1049, March 1988.

[16] K. Kant, "Modelling interprocess communications in distributed programs," In *Proc. Int. Workshop of Petri Nets and Performance Models*, pp. 75–83, Madison, Wisc., August 1987.

[17] L. Kleinrock, *Queueing Systems.* Volume 2, John Wiley and Sons, 1976.

[18] M.A. Holliday and M.K. Vernon, "A Generalized Timed Petri Net Model for Performance Analysis," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 12, pp. 1297–1310, December 1987.

[19] M.J. Fontenot, "Software Congestion, Mobile Servers, and the Hyperbolic Model," *IEEE Trans. Software Eng.*, Vol. 15, pp. 947–962, 1989.

[20] N.D. Gammage and L.M. Casey, "The Software Architecture of a Distributed Processing System," In *Proc. 4th Int. Conf. on Distributed Computing Systems*, pp. 414–431, May 1984.

[21] C. Petri *Kommunikation mit Automaten* PhD thesis, University of Bonn, Bonn, West Germany, 1962 (in German).

[22] R.F. Rashid, *From RIG to Accent to Mach: The Evolution of a Network Operating System*,, U.S.A., 1987.

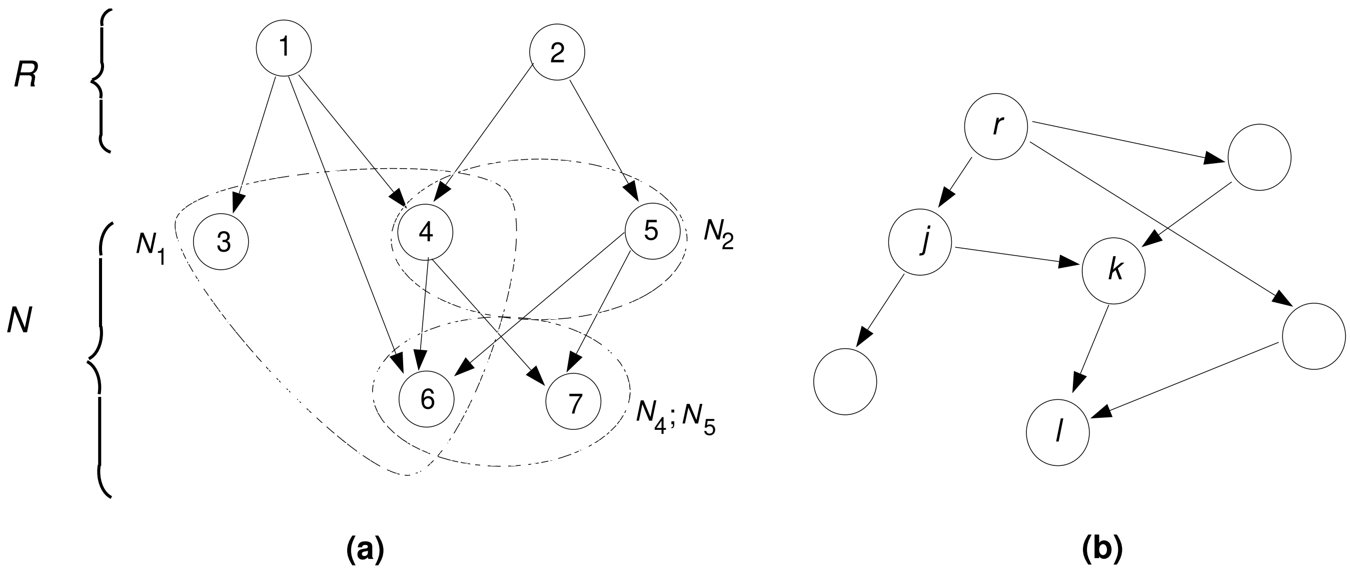

Figure 1: Stochastic Rendezvous Networks
(a) General SRVN Graph
(b) SRVN with Single Reference Task

[23] R.R. Muntz and J.W. Wong, "Asymptotic Properties of Closed Queueing Network Models," In *Proc. Eighth Conf. on Information Sciences and Systems*, pp. 348–352, 1974.

[24] S. Radia, "A Markov Model for Distributed Communicating Sequential Processes," In *Performance '84, Proceedings of the 10th International Symposium on Computer Performance*, pp. 3–16, North Holland, 1984.

[25] S.W. Melvin and Y.N. Patt, "The Use of Microcode Instrumentation for Development, Debugging and Tuning of Operating System Kernels," In *1988 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 207–214, May 1988.

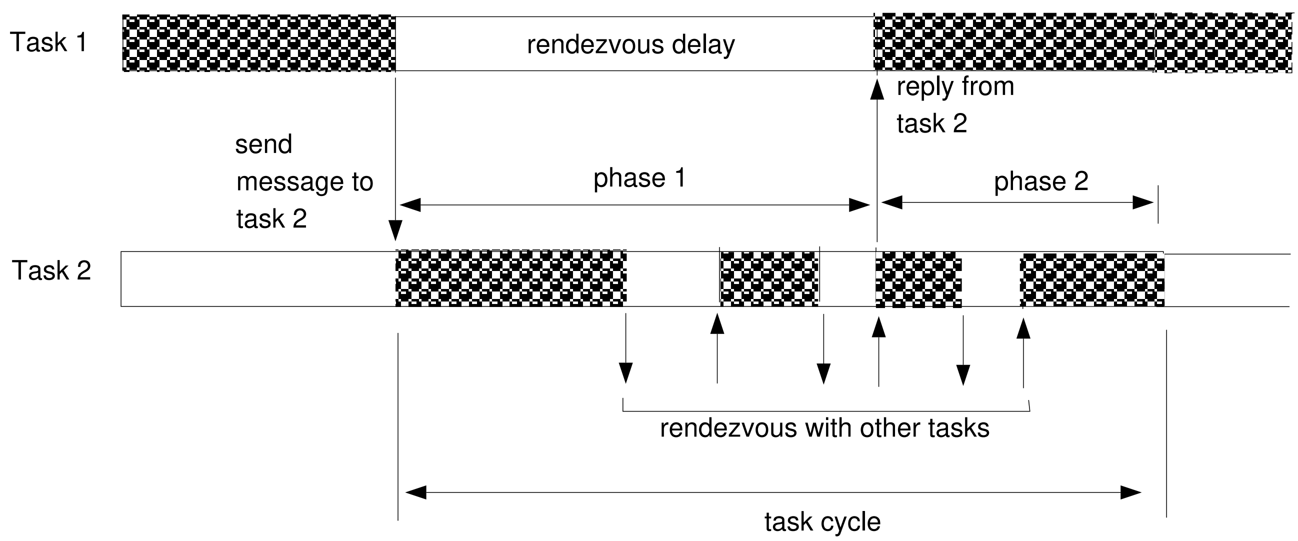

Figure 2: Task Cycles and Phases

Table 3: Tightness of Bounds for the Hierarchical Architecture

Case	S_r	s_{a1}	s_{a2}	s_{n1}	s_{n2}	U_a^+	U_k^+	f_r^+	f_r^-	θ
I	10.0	0.2	0	0	0	0.0784	0	0.098039	0.092593	1.058
	8.0	2.2	0	0	0	0.8628	0	0.098039	0.059524	1.647
	6.0	4.2	0	0	0	1.0000	0	0.059524	0.043860	1.357
	4.0	6.2	0	0	0	1.0000	0	0.040323	0.034722	1.161
	2.0	8.2	0	0	0	1.0000	0	0.030488	0.028736	1.061
	0.2	10.0	0	0	0	1.0000	0	0.02500	0.024876	1.005
II	10.0	0	0.2	0	0	0.0800	0	0.10000	0.092593	1.080
	8.0	0	2.2	0	0	1.0000	0	0.113636	0.059524	1.909
	6.0	0	4.2	0	0	1.0000	0	0.059524	0.043860	1.357
	4.0	0	6.2	0	0	1.0000	0	0.040323	0.034722	1.161
	2.0	0	8.2	0	0	1.0000	0	0.030488	0.028736	1.061
	0.2	0	10.0	0	0	1.0000	0	0.25000	0.024876	1.005
III	10.0	0	0	0.1	0	0.0784	0.039216	0.098039	0.092593	1.058
	8.0	0	0	1.1	0	0.8628	0.431373	0.098039	0.059524	1.647
	6.0	0	0	2.1	0	1.0000	0.5000	0.059524	0.043860	1.357
	4.0	0	0	3.1	0	1.0000	0.5000	0.040323	0.034722	1.161
	2.0	0	0	4.1	0	1.0000	0.5000	0.030488	0.028736	1.061
	0.2	0	0	5.0	0	1.0000	0.5000	0.02500	0.024876	1.005
IV	10.0	0	0	0	0.1	0	0.0400	0.1000	0.092593	1.080
	8.0	0	0	0	1.1	0	0.5500	0.125000	0.059524	2.1000
	6.0	0	0	0	2.1	0	1.0000	0.119048	0.043860	2.714
	4.0	0	0	0	3.1	0	1.0000	0.080645	0.034722	2.323
	2.0	0	0	0	4.1	0	1.0000	0.060976	0.028736	2.122
	0.2	0	0	0	5.0	0	1.0000	0.05000	0.024876	2.010
V	10.0	0.05	0.05	0.025	0.025	0.0594	0.0198	0.099010	0.092593	1.069
	8.0	0.55	0.55	0.275	0.275	0.7253	0.2418	0.109890	0.059524	1.846
	6.0	1.05	1.05	0.525	0.525	1.000	0.3333	0.079365	0.043860	1.809
	4.0	1.55	1.55	0.775	0.775	1.0000	0.3333	0.053763	0.034722	1.548
	2.0	2.05	2.05	1.025	1.025	1.0000	0.3333	0.046500	0.028736	1.414
	0.2	2.5	2.5	1.25	1.25	1.0000	0.3333	0.033333	0.024876	1.340

Table 4: The Effect of Processor Contention

Total No. of Processors (H)	f_r^+	f_r^-	θ
1	0.0245098	0.014925	1.6421
2	0.03125	0.0245098	1.2750
3	0.0625	0.0403226	1.5499
5	0.10990	0.0595238	1.8463

```
repeat forever
     begin
          compute;
          j:= selected receiver task for rendezvous;
          send (j, message, reply);
          {blocked until reply arrives}
     end.
```

(a)

```
repeat forever
     begin
          {blocked until message is available}
          repeat {phase 1}
               compute;
               j:= selected receiver task for rendezvous;
               send (j, message, reply);
               {blocked until reply arrives}
          until (end of phase 1);

          repeat {phase 2}
               compute;
               k:= selected receiver task for rendezvous;
               send (k, message, reply);
               {blocked until reply arrives}
          until (end of phase 2);
          receive (sender, message);
     end.
```

(b)

Figure 3: Skeletal Structures of Tasks:(a) Reference Task (b) Non-Reference Task

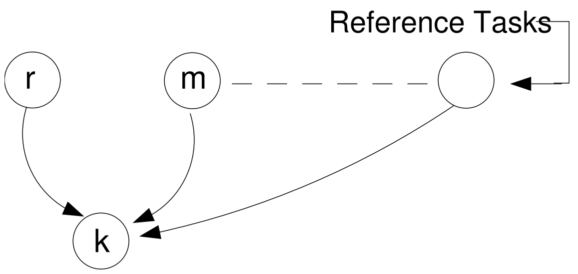

Figure 4: SRVN with no Nested Rendezvous

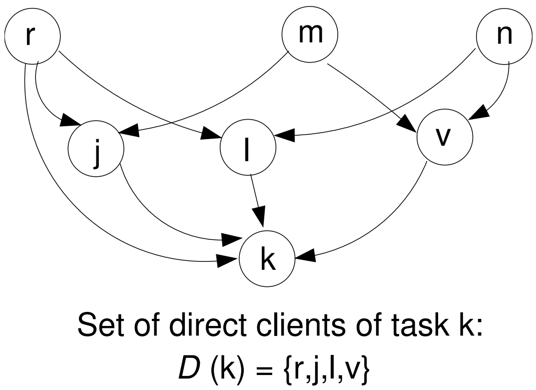

Figure 5: SRVN with Nested Rendezvous

Figure 6: Computation of Bounds with Two Reference Tasks
(a) SRVN Graph
(b) Linear Inequalities for (a)
(c) Linear Inequalities for Symmetrical SRVN
$[S_1 = S_2 = 1.0,\, Y_{13} = Y_{23} = 1.0,\, s_{31} = 1.0,\, s_{32} = 0]$

Figure 7: Example of Box Bounds F_r^-, F_r^+

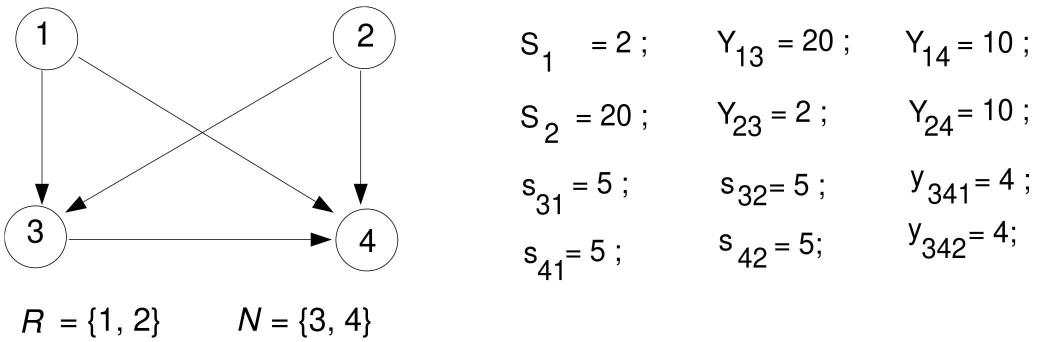

Figure 8: SRVN for the Computation of Box Bounds

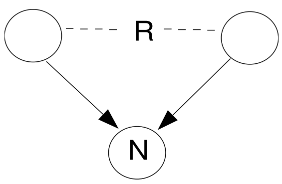

Figure 9: The Client-Server Architecture

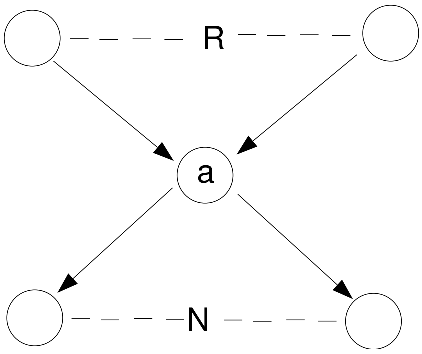

Figure 10: The Hierarchical Architecture

Figure 11: Comparison with Queueing Network Model

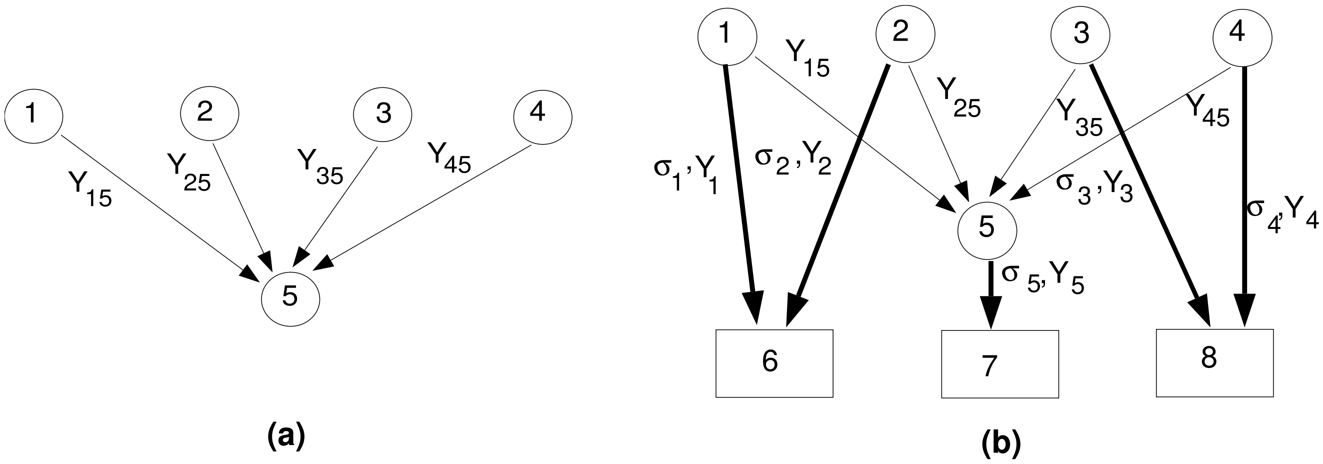

Figure 12: Processor Contention
(a) original SRVN for Client-Server Architecture
(b) modified SRVN with Three Hardware Servers