# Performance analysis of multiprocessor systems in a distributed large data set model using generalised stochastic petri nets

by L Shrihari

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science
Montana State University


Abstract:
A method is proposed for the study of performance characteristics of a 2 or 3 processor homogenous parallel data server system where the individual processors have unique access to statically or dynamically partitioned data in a very large data set using Generalized Stochastic Petri Nets. very large data sets or a warehouse of data can be partitioned and the operations can be done in parallel enhancing their time of completion. Parameters observed in this distributed data environment with multiprocessor architectures have been the effect of mixture of queries that exhibit varying degrees of data parallelism. The scheduling of the mixture of queries is non-adaptive in the case of statically partitioned data and would be adaptive in the case of dynamically partitioned data. The system studied is one in which queries predominate and the information in the servers is updated at specific times where queries are not permitted. Hence, serializability of transactions - as most of them are queries - is not part of this model. A simulator was written to study the Petri Net models and the performability of the different nets with different mixtures of parallel queries. The results clearly show that the architecture chosen is scalable as the data grows and does show the necessity on the part of the analyst to partition data on a timely basis to enhance performance.

# PERFORMANCE ANALYSIS OF MULTIPROCESSOR SYSTEMS IN A DISTRIBUTED LARGE DATA SET MODEL USING GENERALISED STOCHASTIC PETRI NETS.

by L. Shrihari

A thesis submitted in partial fulfillment of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

September 1995

APPROVAL

of a thesis submitted by

L Shrihari

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

9/25/95
Date

Chairperson, Graduate Committee

Approved for the Major Department

9/25/95
Date

Head, Major Department

Approved for the College of Graduate Studies

11/17/95
Date

Graduate Dean

# STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed by the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature Yhunani

Date Sep 25 '95

To my teachers with thanks.

TABLE OF CONTENTS

# LIST OF FIGURES

LIST OF FIGURES (continued)

Figure Page

## ABSTRACT

A method is proposed for the study of performance characteristics of a 2 or 3 processor homogenous parallel data server system where the individual processors have unique access to statically or dynamically partitioned data in a very large data set using Generalized Stochastic Petri Nets. Very large data sets or a warehouse of data can be partitioned and the operations can be done in parallel enhancing their time of completion. Parameters observed in this distributed data environment with multiprocessor architectures have been the effect of mixture of queries that exhibit varying degrees of data parallelism. The scheduling of the mixture of queries is non-adaptive in the case of statically partitioned data and would be adaptive in the case of dynamically partitioned data. The system studied is one in which queries predominate and the information in the servers is updated at specific times where queries are not permitted. Hence, serializability of transactions - as most of them are queries - is not part of this model. A simulator was written to study the Petri Net models and the performability of the different nets with different mixtures of parallel queries. The results clearly show that the architecture chosen is scalable as the data grows and does show the necessity on the part of the analyst to partition data on a timely basis to enhance performance.

# CHAPTER 1

# INTRODUCTION

## Motivation

Most real world applications and the data associated with them have special characteristics that warrant fine tuning in logical and/or physical design of the data store to get speedier answers to queries. Proper logical design such as assignment of relationships through foreign keys or indexes, unique indexes and constraints that tend to model domains enhance performance in most database management systems (DBMS); such gains cannot be significant in a very large data set or data warehouse.

Data warehousing refers to the collection, manipulation, distribution and information processing of very large amounts of data. The data store or database could be from many gigabytes to a few terabytes. Data of this magnitude is usually historical in nature; it was collected over a period of time. Decision Support Systems (DSS), where the knowledge of an enterprise resides, are usually data warehouses that span a large period of time, usually years. With very large amounts of data the warehouse could be a good prognosticator of trends in various meta-models such as retailing, lottery trends, employment statistics, consumer trends, and insurance and actuarial systems. Retrospective analysis of how a business was run and the clues on the profitability of long

range decisions will yield insights into the causal effect of many decisions. Companies and institutions have been waiting for cheaper technologies that could harness all their non-operational data -- data that is not used for day to day operations -- into information. The cost of such information, that is, a transformation of historic data into useful knowledge, is usually very high. Special proprietary systems exist that are prohibitively expensive, leading to the high cost of processing information.

The advent of inexpensive hardware configurations in the form of ever more powerful processors, cheaper and faster memory and media, the declining price to performance ratios, and the vendors' quest to support open and non-proprietary operating systems, database systems, and tools, are important reasons to look into data-warehousing solutions for DSS.

## Limitations Of The Uniprocessor Model

The uniprocessor approach of sequential execution has been the most widely used architecture for database solutions. The performance of this model is limited to the speed of the processor and the usual disk I/O time. The limitations are:

- The uniprocessor architecture is unsuited for a large data set that has data widely spread over time. The data has latent parallelism that is unused by the uniprocessor model.
- The relational database systems tuned to the multiprocessor approach does lead to more parallelism based on the query.

- *Intraquery parallelism*, where the parallel execution of many sub-queries is possible for the same main query.
- *Interquery parallelism* allows the parallel execution of many queries.

The alternatives to the disk bound nature of the uniprocessor model was explored by having the whole database in main memory. This is not only very expensive but also is unfeasible for databases that run in the gigabyte range. This also leads to unstable main memory that as exposed by Leland and Roome [2]. The difference in speed between main memory and other media such as magnetic disks continues to be many orders of magnitude. This I/O bottleneck problem can be improved, if not eliminated, in a relational database system by using many disks of smaller size but higher speed. The asynchronous disk I/O requests available in many UNIX systems would use the different disk I/O controllers and their caches to speed up data retrieval from disk to processor memory.

## Architectures Of Parallel Data Servers

Parallel data servers of the multiprocessor kind can be classified into two major categories - shared nothing and shared everything [11]. In shared everything architecture, the components of the multiprocessor such as processors, main memories, and the many data disks, are connected using a fast interconnection. Figure 1 illustrates the shared everything architecture that is similar to the crossbar switch wherein all the processors have access to all the memories and the disks. In a shared nothing architecture, each

processor has exclusive access to memories and disks. Figure 2 shows the shared nothing model that is similar to the switch based multiprocessor.

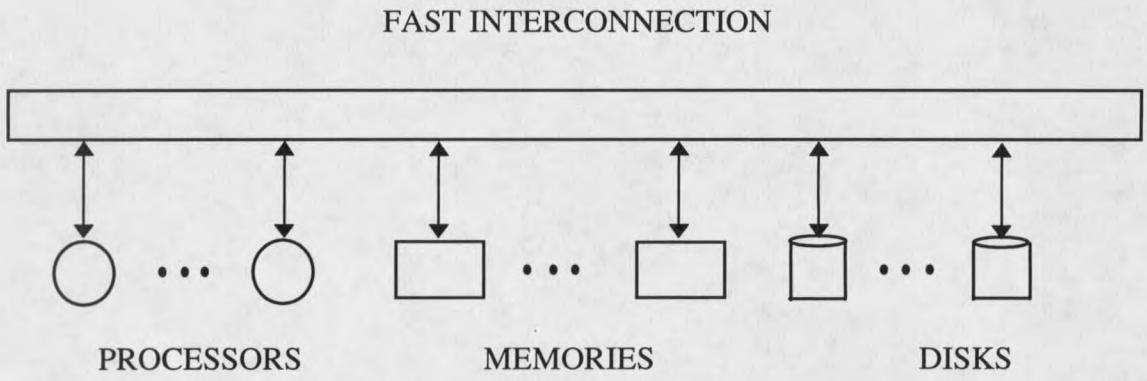


Figure 1. Shared Everything Architecture.

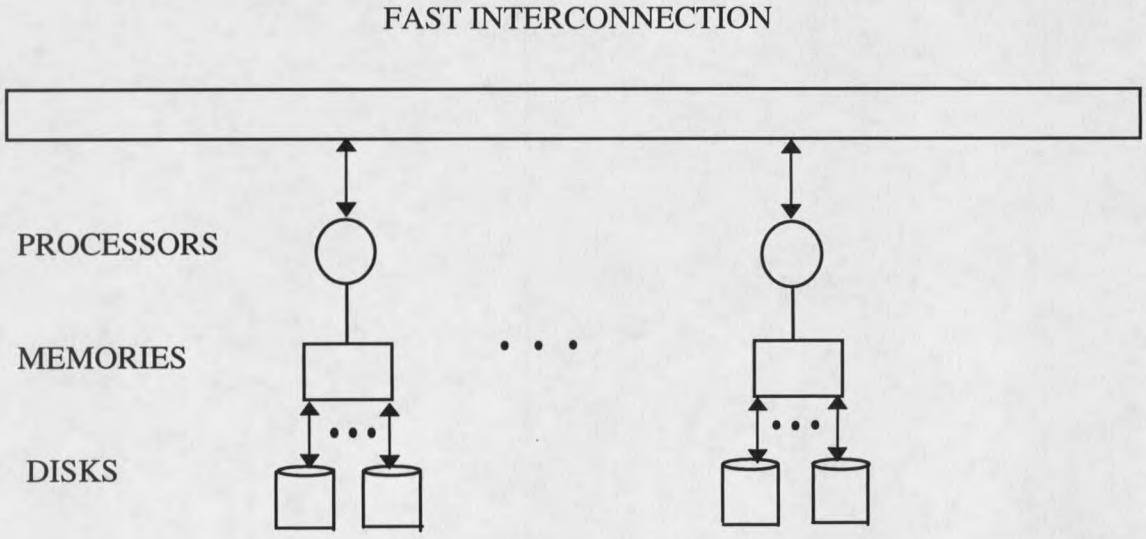


Figure 2. Shared Nothing Architecture.

## DBMS strategies for Parallel Data Servers

Traditionally, shared nothing models are associated with static partitioning of data, and the shared everything models, with dynamic partitioning of data. Static partitioning entails the fragmentation of data across multiple disks that are available to only one processor at all times. Query decomposition is hence dependent on where the data resides and not on the processor availability. This leads to the need for the system analyst to re-partition data that is contended by more user queries on a continual basis, to avoid unbalanced parallelism. However, this model is very extensible; the growth in data can be supported by growth in the number of processors. Examples of shared nothing servers are Tandem NonStop SQL and Teradata DBC/1012.

In dynamic partitioning -- associated with shared everything models -- there is no need for repartitioning data. The processors have access to all the disks and the available processors deal with the next request and have access to requests. In this model the requests made by users do not wait for the appropriate processor that has a hold on the data, but make do with the processor that is available. Efficient usage of CPU and the lack of need to anticipate the partitioning methods are the primary advantages of this model. Examples of shared everything servers are machines from Sequent, data servers from XPRS and the SABRE data server.

## Stochastic Modeling

To measure performance of a system the designers can perform measurement tests on the actual system or use prototypes which emulate the system under different conditions and artificial workloads. Both of these are very involved tasks requiring the design to be mature or complete. The availability of a system or a complete design both at the hardware and software levels is important for measurement testing or for prototyping the system. These conditions are not possible in most real projects.

The study of the behavior and performance of a system can be done in an easier fashion by simulation modeling or through analytical methods. Such studies could be either deterministic or probabilistic. Simulation models can be constructed through modeling tools or by writing programs using a high level language. Analytical models are mathematical descriptions of the behavior of the system. For both of these to be successful the designers need to specify a level of abstraction at all levels of the system -- hardware, software, and applications.

Simulation modeling using probabilistic estimates of the behavior is a possible abstraction of a complex system. Such broad assumptions could allow the designers to focus on the significant aspects of the system omitting the various minute details, which lead to unnecessary complexity in the model. Also, when the level of abstraction is adequate to represent the system, it is easier to observe the differences with changes in the overall design of the system.

Generalized Stochastic Petri Nets provide a sound and convenient method to analyze and validate design approaches. A number of commercial packages exist that have easy-to-use graphical user interfaces that model stochastic petri nets.

# CHAPTER 2

## Petri Nets -- Background

Petri Nets (PNs) or place-transition nets, are bipartite graphs first proposed in 1962 by Carl A. Petri. They are methods for modeling concurrency, nondeterminism, synchronization and control flow in systems. What follows is a cursory introduction; for detailed definitions and rigorous mathematical proofs please refer to [12, 13].

## Definition -- Petri Net

A PN is a set of *places* P, a set of *transitions* T, and a set of directed *arcs* A. The graphical schematic representation of PNs is places as circles and transitions as bars (horizontal or vertical lines). Transitions are connected to and from places by directed arcs. Black dots within a place are called *tokens*. The *state* of a PN is the number of tokens in each place, called a *marking*. The definition of a PN is not complete without the specification of the initial marking $M_0$.

Formally, a PN is a quintuple ( P, T, INPUT, OUTPUT, $M_0$ ) where

$P = \{ p_1, p_2, p_3, \ldots, p_n \}$ is the set of n places.

$T = \{ t_1, t_2, t_3, \ldots, t_m \}$ is the set of m transitions.

$P \cup T \neq \phi$ and $P \cap T = \phi$

INPUT : $( P \times T ) \rightarrow PT_i$ is an input function that describes directed arcs from places to transitions.

OUTPUT : $( P \times T ) \rightarrow TP_i$ is an output function that describes directed arcs from transitions to places.

$M_0 = \{ m_1', m_2', m_3', \ldots, m_n' \}$ is the initial marking.

A transition of a PN is said to be *enabled* if all the input places contain at least one token, so that enabled transitions can fire. Transition firing is accomplished by removing a token from all the input places and placing a token in all the output places. Transitions alter the current marking of the PN yielding a new one. Figures 3 and 4 show the 'before' and 'after' image of the enabling of transition t1. One token is removed from places p1 and p2 and a token is deposited in place p3.

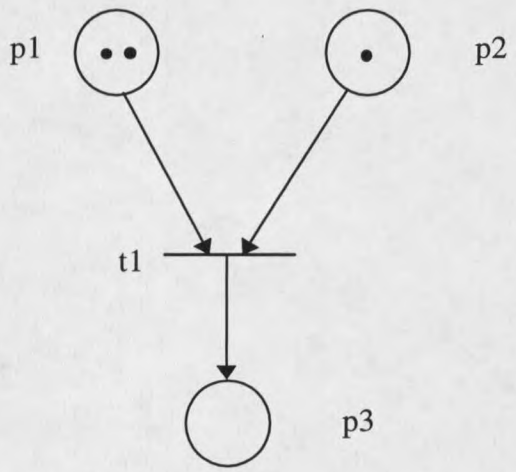


Figure 3 Before transition t1 fires

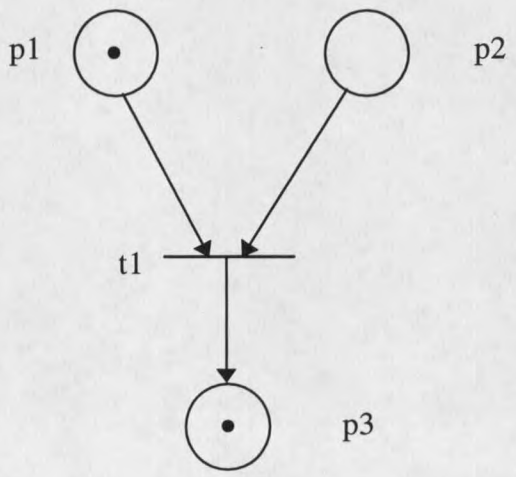


Figure 4 After transition t1 fires in Figure 3

Formally, a transition $t_j$ is said to be enabled in a marking M if

$M(p_i) \geq INPUT(p_i, t_j)$ for all $p_i \in$ set of all places from whom a directed incoming arc reaches $t_j$.

An enabled transition $t_j$ can fire at any time. When a transition $t_j$ is enabled in a marking $M_1$, it fires, resulting in a new marking $M_2$, according to the equation

$$M_2(p_i) = M_1(p_i) + OUTPUT(p_i,t_j) - INPUT(p_i,t_j) \text{ for all } p_i \in P \quad \ldots (1)$$

Marking $M_2$ is reachable from $M_1$ when transition $t_j$ is fired. Every marking is considered to be reachable from itself by enabling no transition. If some marking $M_j$ is reachable from $M_i$ and $M_k$ is reachable from $M_j$, then it follows that $M_k$ is reachable from $M_i$. Reachability of markings exhibit transitive and reflexive relations. The set of all markings reachable from the initial marking $M_0$ is the reachability set $R(M_0)$.

## Definition - Inhibitor Net

An *inhibitor net* is a 6-tuple ( P, T, INPUT, OUTPUT, INHIB , $M_0$) where ( P, T, INPUT, OUTPUT , $M_0$ ) is a PN and

$$INHIB : ( P \times T ) \rightarrow \{0, 1\}$$

is an inhibitor function. In an inhibitor net, a transition $t_j$ is enabled in a marking M if

$M(p_i) \geq INPUT ( p_i, t_j )$ for all $p_i \in$ { set of all places from whom an incoming arc reaches $t_j$ }, and

$M(p_i) = 0$ for all $p_i$ for which $INHIB( p_i, t_j ) \neq 0$.

On firing $t_j$, the new marking is obtained from equation (1).

## Power of Petri Nets

Common activities that can be modeled are concurrency, decision making, synchronization and parallelism, priorities, conflicts and confusion. PN constructs allow representation of many different activities that are not easily represented in finite state machines, such as concurrency and synchronization. Marked graphs can model concurrency and synchronization but cannot model conflicts and merging. The representational power of the PNs is best understood graphically. The most common primitives are given below with their corresponding figures.

*Sequential Execution*: In Figure 5, a precedence relationship exists where transition t2 cannot fire until transition t1 had been enabled and a token was deposited in place p2.

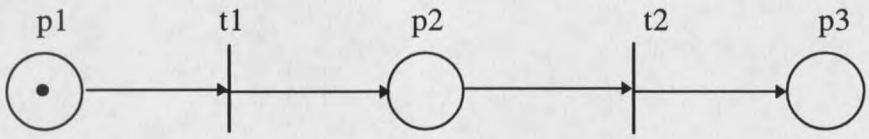


Figure 5. Sequential Execution.

*Concurrency*: In Figure 6, transitions t2 and t3 are concurrent. A transition needs to fork to deposit one or more tokens in more than one place to lead to concurrency.

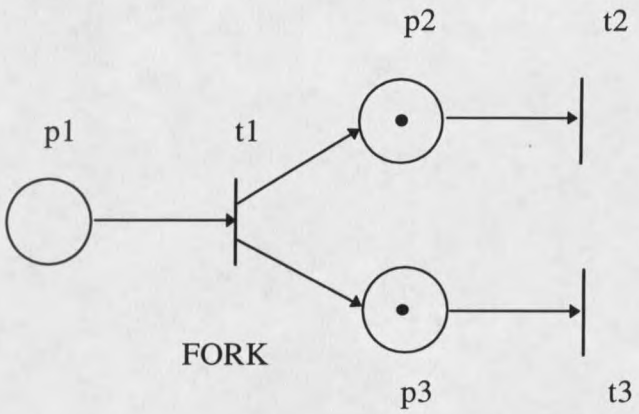


Figure 6. Concurrency.

*Conflict*: In Figure 7, transitions t1, t2 and t3 are in conflict. All transitions are enabled, leading to nondeterminism. This is common in systems where a condition or activity leads to a choice. This situation could be resolved by the assignment of probabilities for the transitions t1, t2 and t3.

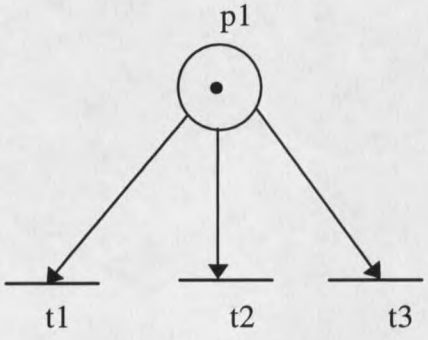


Figure 7. Conflict.

*Merging*: In Figure 8 transitions t1 and t2 merge. This is a classic join operation where different activities lead to the same state or condition. Together with the fork and join it would be easy to model the parbegin-parend construct of Djikstra.

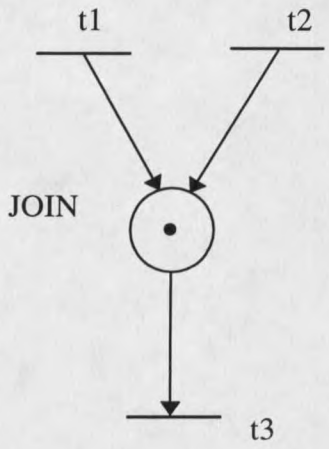


Figure 8. Merging.

*Synchronization:* When the culmination of one or more events in a system is to be waited on then we need a synchronization construct. Figure 9 shows the need to wait until place p1 gets a token. Transition t1 will not be enabled until the token arrives in place p1.

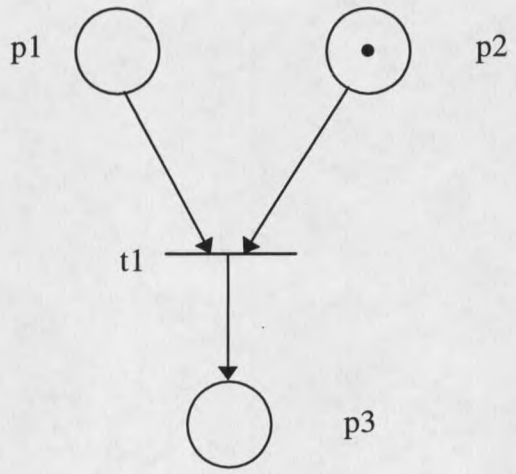


Figure 9. Synchronization.

*Priorities/Inhibitions:* The classical PN definition as given by Petri cannot model priorities. However the extended definition of the inhibitor net will allow priorities. An inhibition is marked from a place to a transition not as a directed arc but as a line that ends with a bulb at the transition.

Figure 10 shows a transition that inhibits the enabling of the transition if there is a token in place p2. In the illustration the transition to be enabled is t1. If there was no token in p2 then the PN results in conflict that must be resolved by a switching distribution in a probabilistic manner.

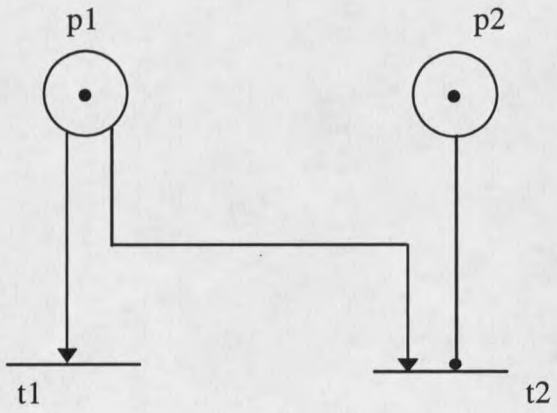


Figure 10. Priorities/Inhibitions.

Inhibitor arcs enhance the modeling power of the PN model. It has been shown that PNs with inhibitor arcs are equivalent to Turing machines [12]. It has also been proven that the original PNs are less powerful than Turing machines and can only generate a proper subset of context-sensitive languages.

Classical PNs are useful in investigating qualitative or logical properties of concurrent systems, such as safeness, resource allocation or conservation, mutual exclusion, absence of deadlocks or their existence, reachability and study of the states of the system. They are, however, not very suitable for quantitative analysis of systems.

## Stochastic Petri Nets

To enhance the quantitative features of PNs the concept of time was needed in the definition of PNs. The idea of introducing time into a PN is based on the idea that for a system model in a given marking (state), a certain amount of time must have elapsed before an event occurs. The result of some activity in a given state is thus tracked by time. Ramamoorthy and Ho [11] investigated the use of *Timed Petri Nets* (TPN)in which places or transitions were incorporated with *deterministic* (constant) time delays. The analysis of such TPNs was found intractable in most cases except where the PN was equivalent to a marked graph.

The doctoral dissertations of S. Natkin [9] and M. K. Molloy [8] independently led to *Stochastic Petri Nets* (SPNs), where they associated exponentially distributed *random* delays to transitions. These methods are isomorphic to Continuous Time Markov Chains (CTMCs). Each marking of an SPN is equivalent to a state of the CTMC. SPNs suffered from graphical and analytical complexity as the systems to be modeled increased in size and complexity. The number of states of the CTMC along with the SPN reachability set exploded, rendering SPNs to be used for modeling systems of smaller size and complexity.

## Definition - SPN

An SPN is a 6-tuple ( P, T, INPUT, OUTPUT, $M_0$, F ) where ( P, T, INPUT, OUTPUT, $M_0$ ) is the marked untimed PN and F is a function in the domain of the cartesian product of the reachability set and the transitions ( $R[M_0] \times T$ ), which associates with each transition in each reachable state, a random variable.

Every transition has a firing delay associated with its action. This is the time that will elapse before the transition can fire. The firing delay is a random variable with a negative exponential probability distribution function.

## Generalized Stochastic Petri Nets

Generalized Stochastic Petri Nets (GSPNs), proposed by Marsan, Balbo and Conte [3, 4, 5] allow the firing delays to be zero or the transition to be fired immediately. GSPNs are obtained by allowing transitions to belong to two different classes: immediate transitions and exponential transitions. Immediate transitions fire in zero time once they are enabled. Exponential transitions are fired in exponentially distributed firing times. The immediate transitions have priority over timed transitions. It is important to note that associating no time with some transitions enhances the qualitative modeling of a system. Time is associated with only those transitions that have a larger (time-wise) impact in the modeled system. The availability of a logical structure that can be used along with the

timed activities allows the construction of elegant performance models of complex systems.

Formally, a GSPN is an 8-tuple ( P, T, INPUT, OUTPUT, INHIB, $M_0$, F, S ) where

- ( P, T, INPUT, OUTPUT, INHIB, $M_0$ ) is an inhibitor-marked PN,
- T is partitioned into two sets: $T_{IMM}$ of immediate transitions and $T_{EXP}$ of exponential transitions,
- F: ( $R[M_0] \times T_{EXP}$ ) $\rightarrow R$ is a firing function associated with each $t \in T_{EXP}$ each $M \in R[M_0]$, an negative exponential random variable with rate F ( M, t ),
- Each $t \in T_{IMM}$ has zero firing time in all reachable markings, and
- S is a set, possible empty, of elements called random switches, which associate probability distributions to subsets of conflicting immediate transitions.

Exponential transitions are drawn as rectangular (filled or unfilled) bars or segments. All other graphical notations are the same as those of PNs. Figure 11 shows the immediate transition t1 and the exponential transition t2.

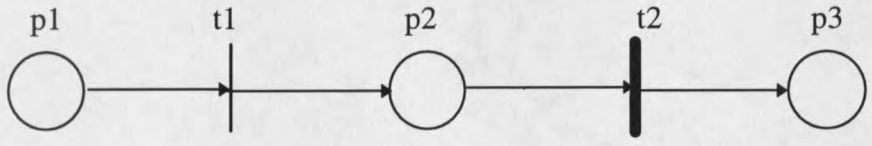


Figure 11. Immediate transition t1 and exponential transition t2.

In a GSPN marking, more than one transition can be enabled at one time. If $M_i$ is a reachable marking and if $T_i$ is the set of enabled transitions in $M_i$, then $T_i$ includes only exponential transitions. The transition $t_j \in T_i$ fires with probability

$$P\{ t_j \} = \frac{F(M_i, t_j)}{\sum_{t_k \in T_{IMM}} F(M_i, t_k)}$$

If $T_i$ consists of only one immediate transition $t_j$, then $t_j$ will fire. If $T_i$ consists of two or more immediate transitions, then a probability mass function is specified on the set of enabled immediate transitions by a member of S. The enabled transition is chosen according to this probability distribution. The set of all immediate transitions, together with the switching distribution of these transitions is called the *random switch.* The probabilities in a switching distribution could be independent of the current marking (*static* random switch) or dependent on the current marking (*dynamic* random switch).

The GSPN model assumes that transitions are fired one by one, even if all the transitions that could be enabled constitute a set of nonconflicting immediate transitions. This simplifies the implementation of these models.

A GSPN marking which enables only timed transitions is called a *tangible* marking and one which enables at least one immediate transition is called a *vanishing* marking. It is imperative on the part of the designer of the net to decide on a semantic policy for the execution of the net [6, 7].

# CHAPTER 3

## METHOD

### GSPN for the 2-processor system.

The GSPN in Figure 12 is a model of the 2-processor system. It has 10 places, 5 immediate transitions and 4 timed transitions. The key phenomenon we want to model is the query request that would be one of two types -- requiring both processes or only one of them. The query coordinator resides in a machine that has the metadata regarding the two machines and the data contained in them. It is assumed that the overhead of the supervisory programs is minimal and the processors are statistically equivalent. The data access requests by a processor and the transmission of the result are assumed to be asynchronous. This is modeled using timed transitions. The fast interconnection is assumed to yield high data throughput. The access delay with this interconnection is assumed to be insignificant in comparison to the time to process the query.

Place p0 is the query coordinator that makes the decision on whether a given job request can be performed by one processor or by two processors in parallel. The input transition rate into p0 is probabilistic with negative exponential probability of mean $\alpha$, much like the Poisson arrival rates in queuing networks. The transitions t1 and t2 constitute a random switch of probabilities $\lambda_1$ and $\lambda_2$. Place p1 signifies that a job request

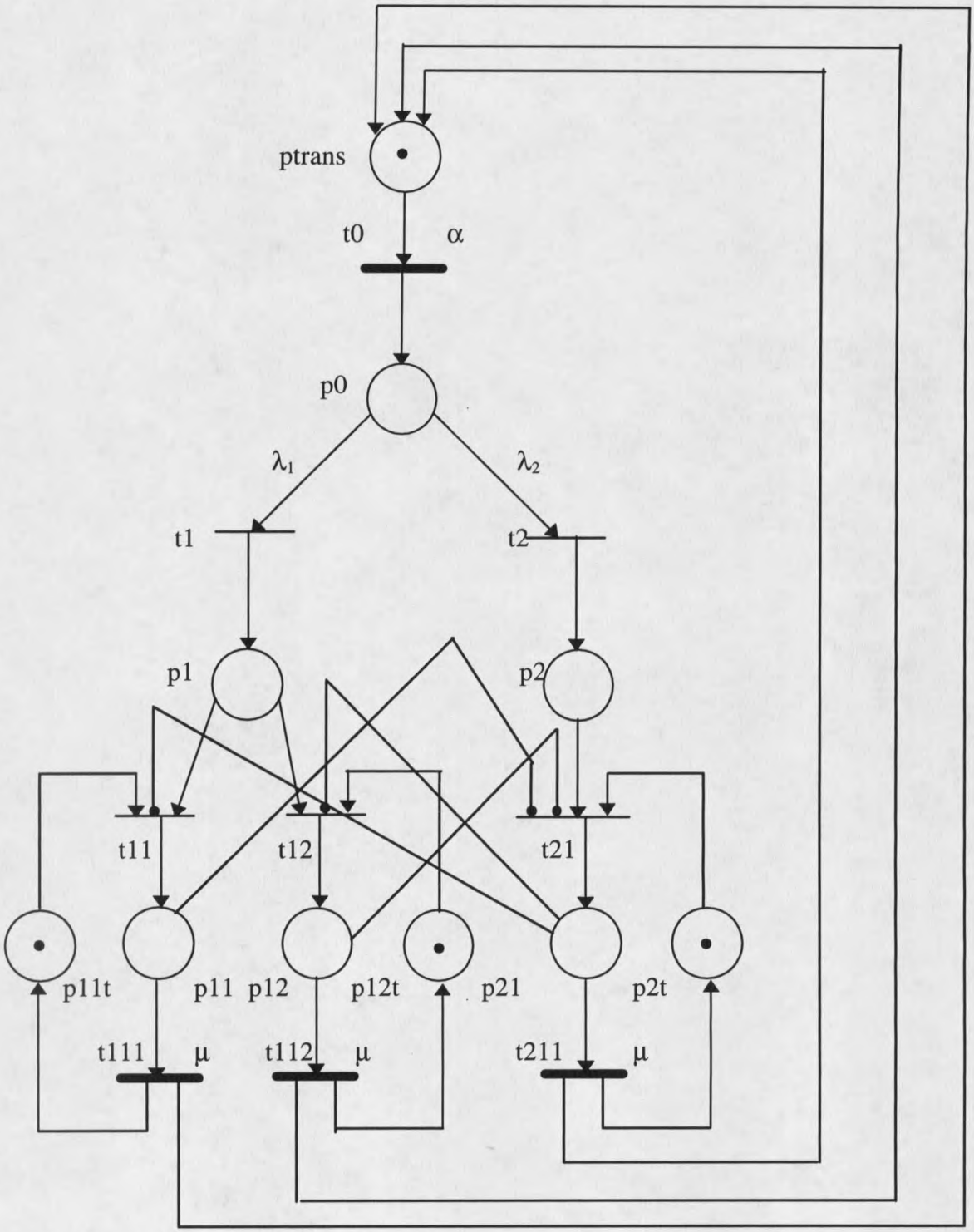


Figure 12. GSPN model of the 2-processor system.

from p0 is a one-processor job and place p2 signifies that a job request from p2 is a 2-processor one.

The associated delay rates $\lambda_1$ and $\lambda_2$ of the random switch mean that place p1 will have probability $( \lambda_1 / ( \lambda_1 + \lambda_2 ))$ of receiving a token or getting a job request from p0, and place p2 will have probability $( \lambda_2 /( \lambda_1 + \lambda_2 ))$. If the sum of the two values is set to one, we would have normalized values for the random switch. The places p11, p12 and p21 are places where the individual processors (in the case of p11 and p12) or both ( in the case of p21) are working in their private memories. The places p11t, p12t and p21t are to signify that the processors are available for processing. So when a token is in p1, the presence of a token in p11t or p12t would mean that both the processors are idle and that either one could be assigned the request. If p12t has no tokens -- processor 2 is busy -- then p11 is the only choice. Place p21 has an input transition which is also inhibited by the places p11 and p12. The 2-processor request cannot go further if both or one of the processors is busy. Transitions t111, t112 and t211 are timed transitions that model the processing times of the job request with a delay rate of mean $\mu$ and they are assumed to be distributed in a exponential distribution. This would be a stochastic approximation of the internal details of database accesses, the processing of data and the transmission occurring through a fast interconnect. The measure of the load factor on the system could be characterized as the ratio of $\alpha$ by $\mu$.

## GSPN for the 3-processor system.

Figures 13, 14 and 15 together constitute the GSPN for the 3-processor model. It has 21 places, 12 immediate transitions and 9 timed transitions. Requests from users of the system are denoted by the exponential transition t1 that arrives from place ptrans to place p0. As in the 2-processor model it is assumed that the overhead of supervisory programs is minimal, the processors are statistically equivalent, and the data access requests by a processor and the transmissions of results are asynchronous.

Place p0 signifies the query coordinator which has to decide on whether the query requires processing by one, two or three processors that work in tandem. The input transition rate t0 into p0 is distributed over a negative exponential probability of mean $\alpha$. The transmissions t1, t2 and t3 form the new random switch of probabilities $\lambda_1$ ,$\lambda_2$ and $\lambda_3$. Places p1, p2 and p3 signify that a job request from the query coordinator (p0) is a one, two or three processor requirement. The delay rates $\lambda_1$ ,$\lambda_2$ and $\lambda$ of the random switch mean that p1 will have probability ( $\lambda_1$ / ( $\lambda_1 + \lambda_2 + \lambda_3$ )) of receiving a token from the query coordinator, and, place p2 and p3 will have ( $\lambda_2$ / ( $\lambda_1 + \lambda_2 + \lambda_3$ )) and ( $\lambda_3$ / ( $\lambda_1 + \lambda_2 + \lambda_3$ )) as probability of receiving a token from the coordinator, respectively. Applying the same principle of normalization of the $\lambda$ values to yield a sum of one we simplify the comparison of the ratios of the different job mixes being processed by the query coordinator.

Transitions t11, t12 and t13 are immediate transitions which are fired when a token exists in p11t or p12t or p13t, and a token exists in place p1, and there are no tokens in places p21, p22, p31, p32 and p33 modeled by the inhibitor arcs. The presence of a token

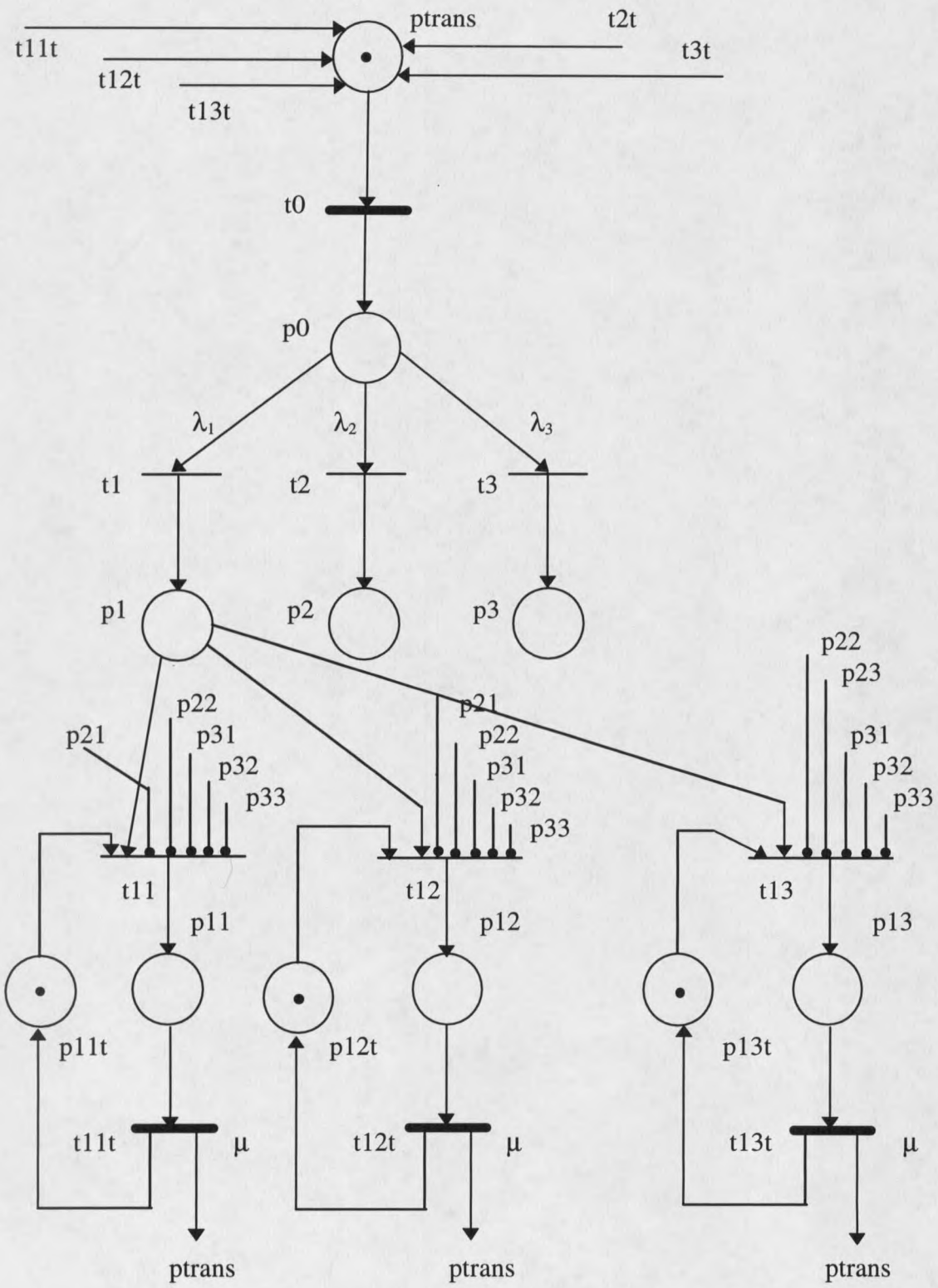


Figure 13. GSPN model of the 3-processor system.

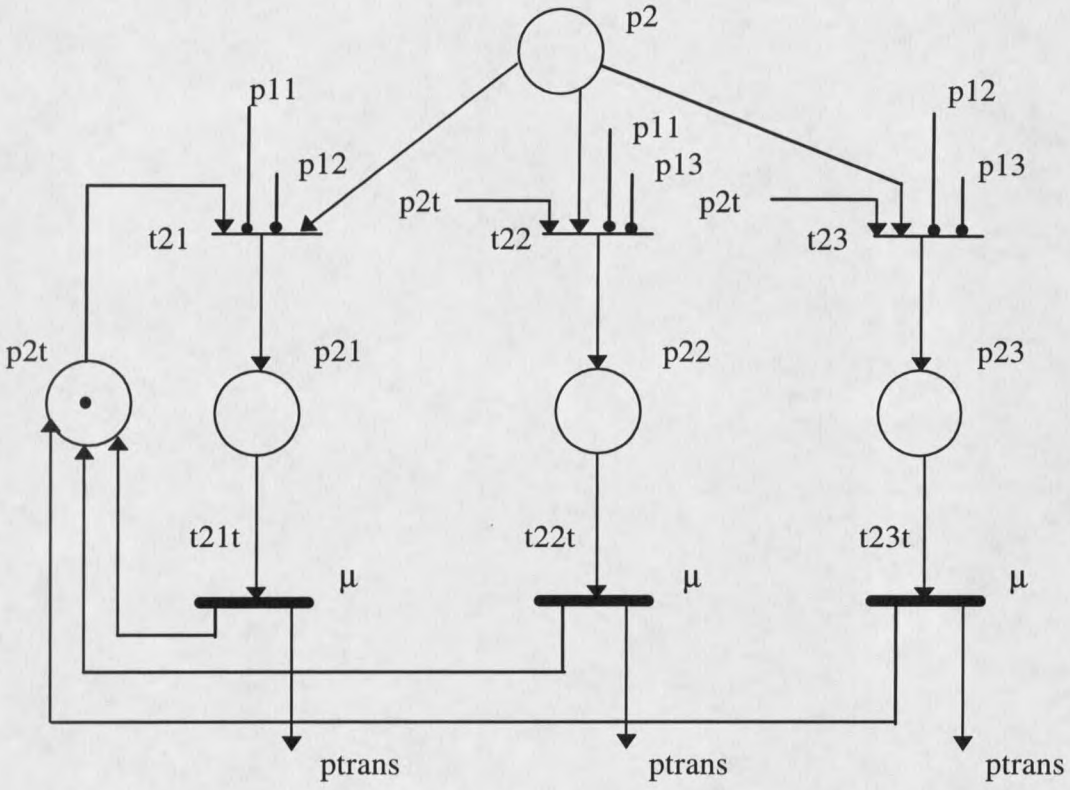


Figure 14. GSPN model of the 3-processor system (continued).

in places p11t, p12t and p13t implies the availability of the individual processors. The timed transitions t11t, t12t and t13t model the time taken by the processors to complete the query request. They have a delay of μ associated with them.

Place p2 could receive a token from p0 based on the random static switch. A token can be placed in place p21 if and only if places p11 and p12 do not have a token in them -- they are the processors that are used for this 2-processor query -- and place p2t has a

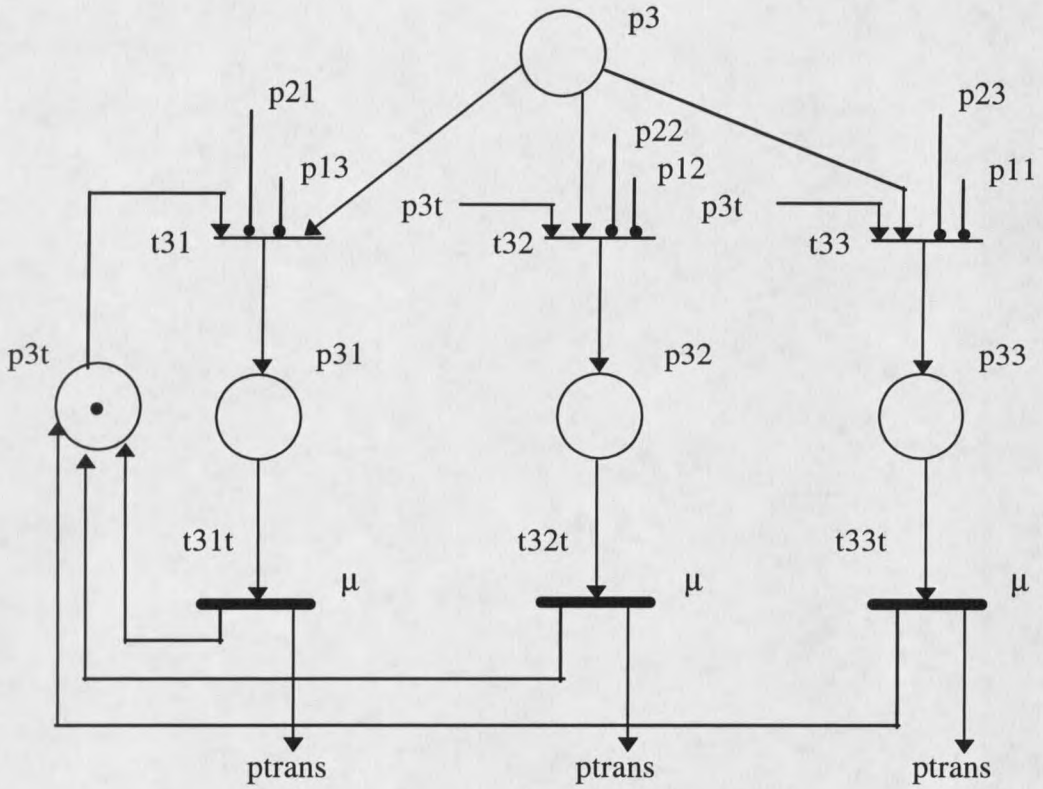


Figure 15. GSPN model of the 3-processor system (continued).

token in it. Since there cannot be more than one 2-processor request in a 3-processor combination at any one time, the absence of a token in p2t will mean that any two processors are busy and no other 2-processor queries can be entertained at that time.

Place p3 in Figure 15 is similar to place p2 in Figure 14, with the essential difference that it models the 3-processor request. A token could be put in place p3 from place p0 by way of the random switch. A token can result in place p31 from place p3 if there are no tokens in places p21 and p13 and there exists a token in place p3t. There

cannot be tokens in places p31, p32 and p33 at the same time as the number of requests through the system are served on a First Come First Served (FCFS) basis. This is characterized by having only one token in the initial marking of the net for place p3t so at any time there can be only one 3-processor request.

## Implementation

Figure 16 illustrates the architecture of the implementation of the GSPNs in Figures 12 through 15. Again the measure of the load factor is characterized by the ratio of $\alpha$ by $\mu$.

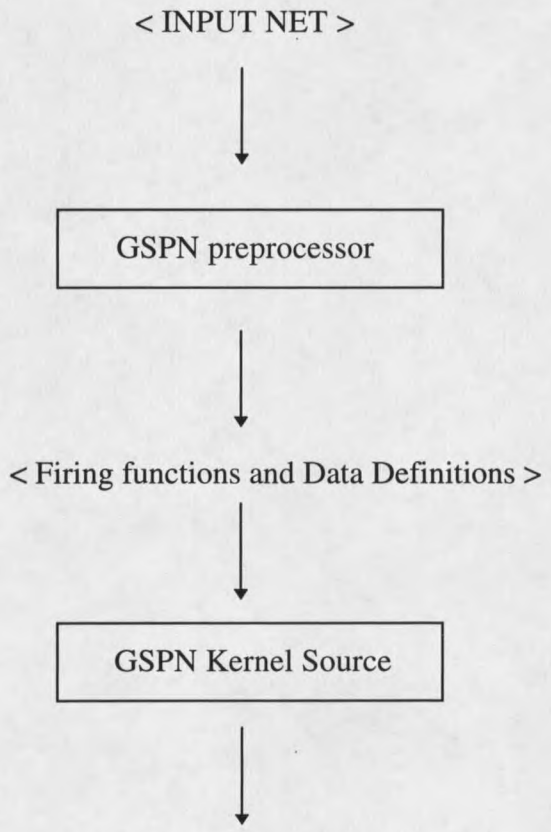


Figure 16. Implementation scheme of the GSPN.

The schematic GSPN is written out in the form of an INPUT-NET using the grammar in Figure 17.

---

```
Firing Functions and Data Definitions  = START  definitions END
definitions  = NULL  | definitions placedefns | definitions transdefns
placedefns  = PLACELIST placelist PLACEEND
transdefns  = TRANSLIST translist TRANSEND
placelist    =  place |  place placelist
place        =  PLACE NAME markings = value
translist    =  transition |  transition translist
transition   =  TRANS name type = value  rate  = value
                   INPUT ilist OUTPUT olist INHIB hlist
ilist        =  NULL  | ilist name
olist        =  NULL  | olist name
hlist        =  NULL  | hlist name
name         =  [A-Za-z][A-Za-z0-9_]*
value        =  [-+.0-9edEd]+
```

---

Figure 17. The grammar for the INPUT-NET.

The reserved words used in the grammar are START, END, PLACELIST, PLACEEND, TRANSLIST, TRANSEND, PLACE, TRANS, INPUT, OUTPUT, INHIB, markings, type, and rate.

The value for MARKS in the place definition is the number of tokens in the place at start (initial marking), and the value of 0 for type in the transition definition means an immediate transition and 1 is a timed transition. The lexical portion of the preprocessor

will allow multi-line or single-line C ( /* */ ) style comments and single-line C++ comments ( // ). The preprocessor was written using the above grammar in lex and yacc.

For the 2-processor model in Figure 12 the INPUT-NET is given in Figure 18.

---

```
START
PLACELIST
        PLACE  p0      markings = 0
        PLACE  p1      markings = 0
        PLACE  p11t    markings = 1
        PLACE  p12t    markings = 1
        PLACE  p11     markings = 0
        PLACE  p12     markings = 0
        PLACE  p2      markings = 0
        PLACE  p2t     markings = 1
        PLACE  p21     markings = 0
        PLACE  ptrans markings = 1
PLACEEND

TRANSLIST

        TRANS  t0    type = 1 rate= 1.0  INPUT ptrans OUTPUT p0 INHIB
        TRANS  t1    type = 0 rate= 0.7  INPUT p0 OUTPUT p1 INHIB
        TRANS  t2    type = 0 rate= 0.3  INPUT p0 OUTPUT p2 INHIB
        TRANS  t11   type = 0 rate= 1.0  INPUT p1 p11t OUTPUT p11 INHIB p21
        TRANS  t12   type = 0 rate= 1.0  INPUT p1 p12t OUTPUT p12 INHIB p21
        TRANS  t111 type = 1 rate= 1.0  INPUT p11 OUTPUT ptrans p11t INHIB
        TRANS  t112 type = 1 rate= 1.0  INPUT p12 OUTPUT ptrans p12t INHIB
        TRANS  t21   type = 0 rate= 1.0  INPUT p2 p2t OUTPUT p21 INHIB p11 p12
        TRANS  t211 type = 1 rate= 1.0  INPUT p21 OUTPUT ptrans p2t INHIB

TRANSEND

END
```

---

Figure 18. INPUT-NET for GSPN in Figure 12.

The preprocessor will read the INPUT-NET, parse the contents of the net and create the transition definitions needed for the kernel to operate on according to the method in Figure 19.

---

- Insure that every place has an input and an output transition associated with it.
- Set the place definition structure with the initial marking.
- Set the transition definition structure with the type of transition and the rate values; every transition will have a firing function associated with it.
- // Write the transition firing functions for all transitions

```
for all transitions do
  for all  places in the input list of the transition
    for all transitions that are in the input list of the place
      remove a token from the place and set these transitions as vanishing
      and remove them from queue.
    endfor
    for all transitions that are in the inhib list of the place
      disable the transition and remove from the queue.
    endfor
  endfor
  for all places  in the output list of the transition
    for all transitions that are in the inhib list of the place
      add a token to the place and add the transition to the queue.
    endfor
    for all transitions that are in the input list of the place
       add transition to queue.
    endfor
  endfor
endfor
```

---

Figure 19. The preprocessor algorithm.

The kernel source is bound (linked) with the output from the preprocessor to produce a GSPN executable instance for the GSPN INPUT-NET. The core of the kernel is based on the algorithm in Figure 20.

---

- From all immediate transitions add the probabilities as the sum of the rates.
- For all the immediate transitions recompute the normalized probabilities as a ratio of the normalized sum to the delay (inverse of rate)
- Initialize the place and transition data-structures.
- loop

  //

  // The transition update functions created at the preprocessing stage will
  // contain the next transition to follow if any - immediate and/or
  // exponential transition(s). In the case of the exponential transition the
  // delay is computed through a random negative exponential function.

  //

  - If an immediate transition is in the queue execute the transition by firing the update function created for it in the pre-processor stage. If another immediate transition exists then execute it. Continue to exhaust every immediate ransition. The time elapsed will be increased by the delay.
  - If no immediate transition exists then choose one randomly.
  - Find whether a timed transition exists. If none does then the petri net has been deadlocked. If a timed transition exists fire.
- until the number of times to transit is done.

---

Figure 20. The algorithm for the kernel.

## Results

The measure of performance of the processor models is the steady state average number of processors that are active. That is the sum of probabilities in all markings where

there are tokens in places p1 and p2 in the 2-processor model and in all markings where there are tokens in places p1, p2, and p3 in the 3-processor model. The load factor is characterized as the ratio of $\alpha$ by $\mu$.

The Figures 21 and 22 are charts that show performance measure with respect to the mixture of the job stream and the load factor for the 2-processor model. In Figure 21 we can see the improved performance as the job mix goes from 90% 1-processor queries and 10% 2-processor queries to the opposite side of the spectrum which is 10% 1-processor queries and 90% 2-processor queries. In Figure 22 as the load increases on the two systems we can observe the uniform degradation of the system.

The results for the 2-processor model clearly suggests that as the load increases the performance of the system will degrade irrespective of the job mix. Also, the larger the fraction of the 2-processor queries, the better the performance. It is important to realize that one processor queries do contend for the same processor in this model.

Figure 23 is the chart that shows the effect on performance measure as the mixture of the different queries presented to the system is altered in the 3-processor model. Again, it is evident that the larger the proportion of 3-, 2-, and 1-processor requests, in that order, the greater the performance of the system. The effect of the load factor on performance is again uniform degradation as the load increases. It was always assumed in the measurement that at no time will there be a lack of the 1- or 2-processor requests in the system. If that was not the case and the only job requests were 3-processor requests, the performance measure will be close to three.

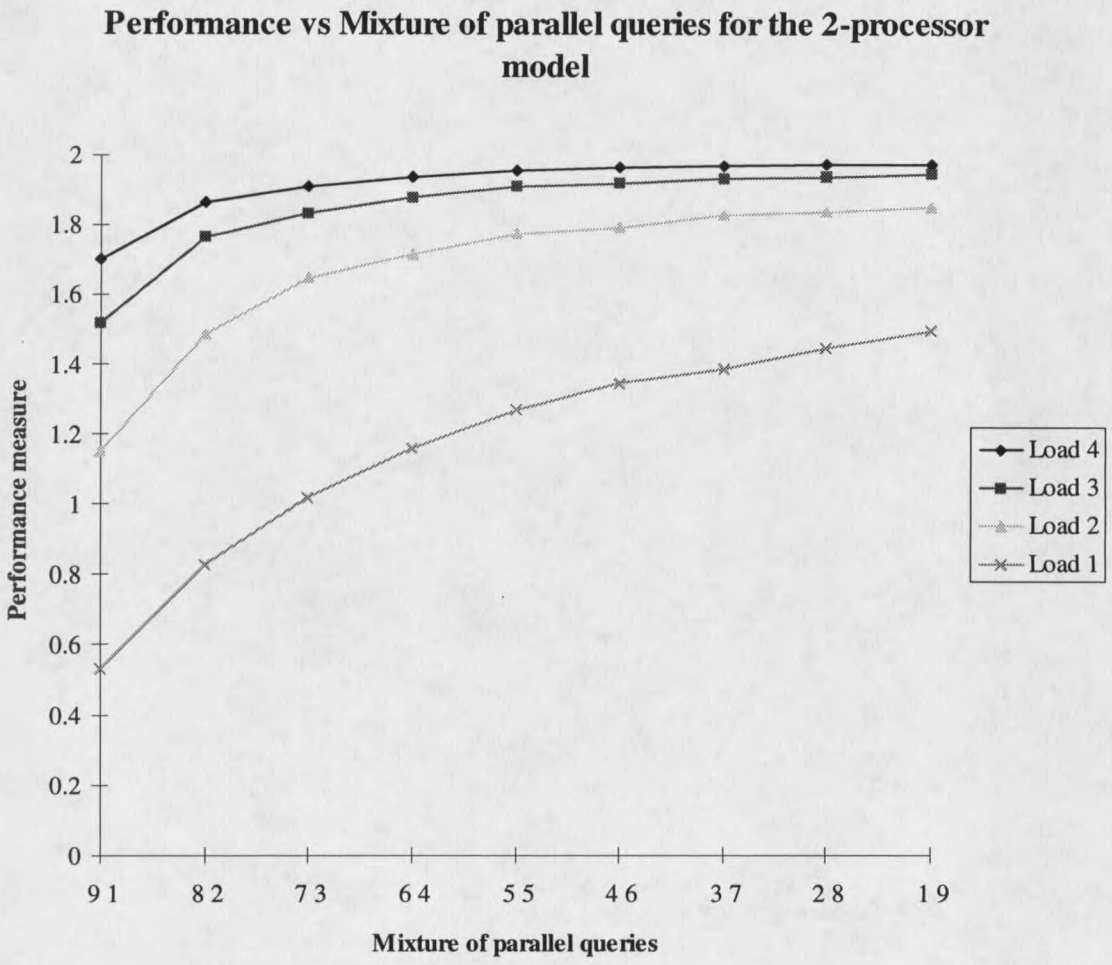


Figure 21. Performance vs Mixture of parallel queries for the 2-processor model.

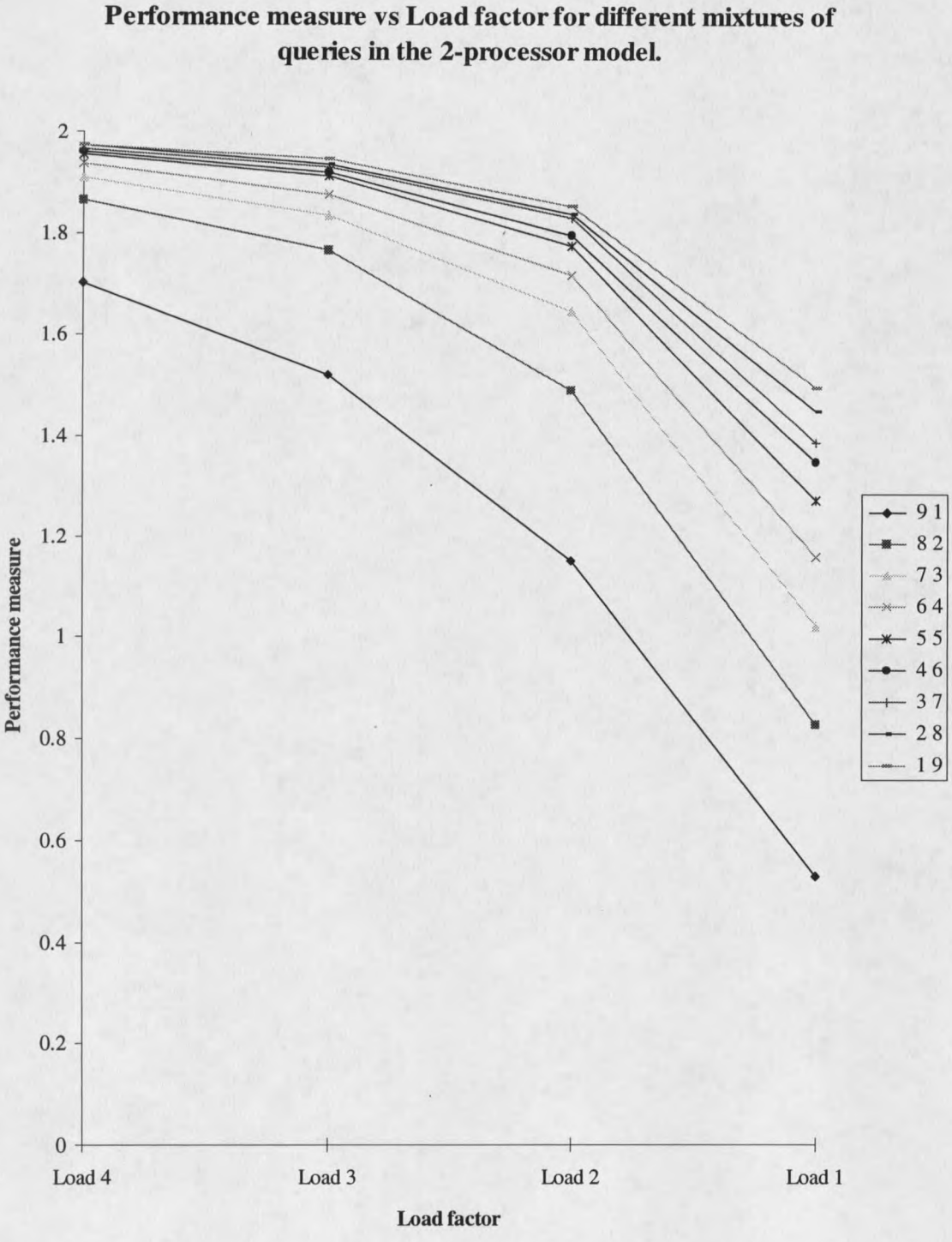


Figure 22. Performance measure vs Load factor for different mixtures of queries in the 2-processor model.

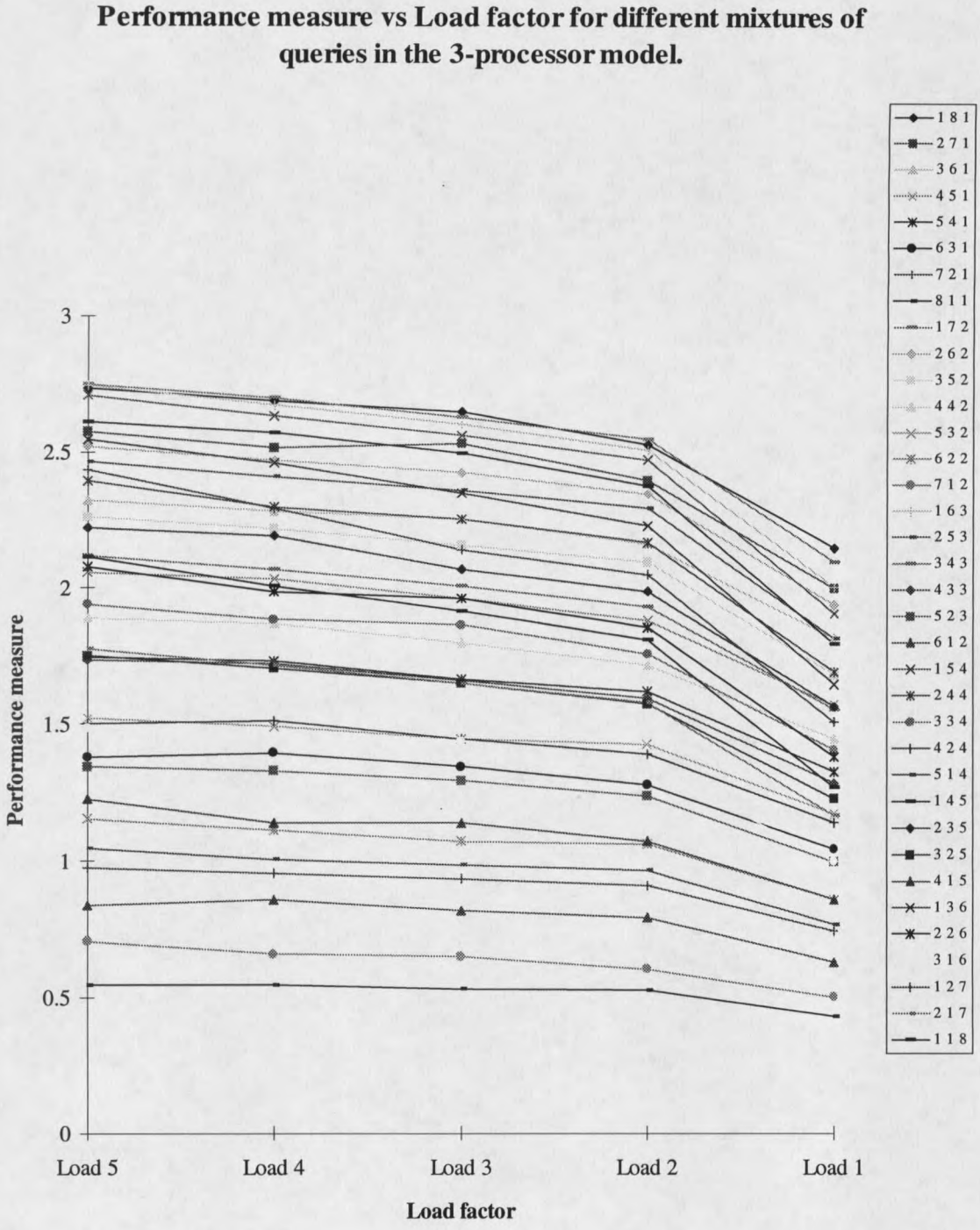


Figure 23. Performance measure vs Load factor for different mixtures of queries in the 3-processor model.

## Limitations of the current models and suggestions for future research

The current models are based on the First Come First Served approach. Most real world solutions are of this approach. It is possible to change the scheduling of the requests by a lookahead mechanism. For example, when a 2-processor request is first in the queue and the next one is another 2-processor request, this lookahead method will search for the next request to see if it is a 1-processor request.

The current models are bulky and the number of places and transitions will increase as the number of processors increase. The net needs to be redone every time a processor is added to the system. A general purpose solution that will model the processors as the number of tokens in a place in the initial marking of the net would be desirable.

The models had a high degree of liveness and never experienced deadlock. It would be desirable to modify the implementation to study these properties.

## BIBLIOGRAPHY

1. 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, pages 1297-1310, Dec 1987. IEEE Computer Society Press.

2. M. D. P. Leland and W. D. Roome. *The Silicon Database Machine.* In Proceedings 4th International Workshop on Database Machines, pages 169-189, March 1985.

3. M. A. Marsan, G. Conte, and G. Balbo. *A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems.* ACM Transactions on Computer Systems, Vol. 2, No. 2, pages 93-122, May 1984.

4. M. A. Marsan. *Stochastic Petri Nets: An Elementary Introduction.* In Gregorz Rozenberg, editor. Advances in Petri Nets 1989, pages 1-29. Springer-Verlag, 1989.

5. M. A. Marsan, G. Balbo, and G. Conte. *Generalized Stochastic Petri nets revisited: Random switches and priorities.* ACM Transaction on Computer Systems. Vol 2 No. 1, pages 93-122, May 1984.

6. M. A. Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. *The Effect of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets.* IEEE Transactions on Software Engineering. Vol. 15 No. 7, pages 832-846, July 1989

7. M. A. Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani.. *On Petri nets with Stochastic Timing.* In Proceedings of the 1985 Workshop on Timed Petri Nets, pages 80-87, Torino, Italy, July 1985. IEEE Computer Society Press.

8. M. K. Molloy. *Performance analysis using stochastic Petri nets.* IEEE Transactions on Computers, Vol. C-31, No. 9, pages 913-917, 1982.

9. M. K. Molloy. *On the integration of delay and throughput measures in distributed processing models.* Ph.D. dissertation, University of California, Los Angeles, 1981.

10. S. Natkin. *Les reseaux de Petri Stochastiques et leur application a l' evaluation des systems informatiques.* Ph.D. dissertation, CNAM-PARIS, June 1980.

11. M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Englewood Cliffs, Prentice-Hall, 1991.

12. J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, Prentice Hall, 1981.

13. C. A. Petri. *Communication with Automata.* Ph.D. dissertation, Tech. Rep. RADC-TR-65-377, Rome Air Development Center, Rome, NY, 1966.

14. C. V. Ramamoorthy and G. S. Ho. *Performance evaluation of asynchronous concurrent systems using Petri nets.* IEEE Transactions on Software Engineering, Vol SE-6, No. 5, pages 440-449, Sep 1980.