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.

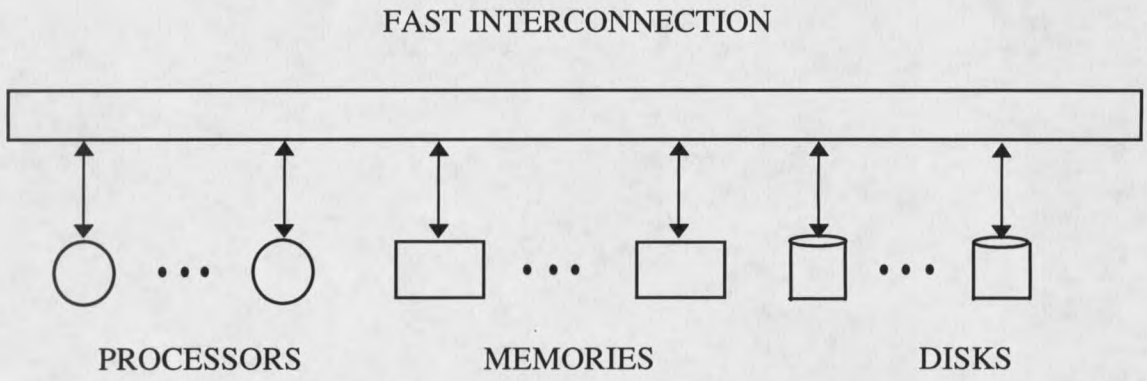

Figure 1. Shared Everything Architecture.

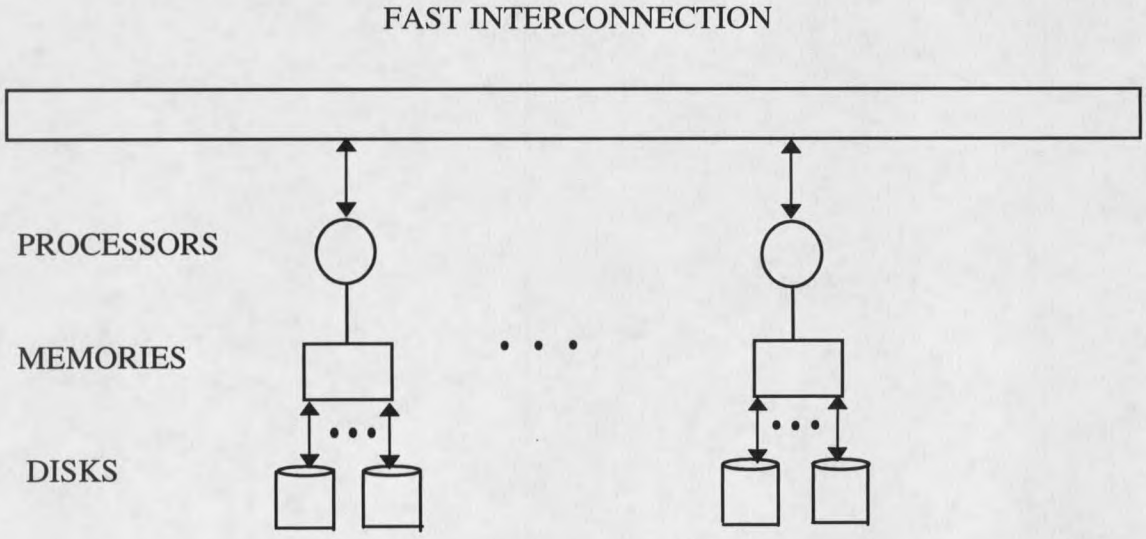

Figure 2. Shared Nothing Architecture.

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.

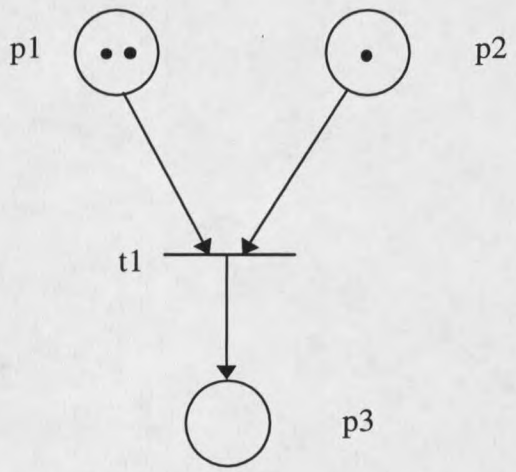

Figure 3 Before transition t1 fires

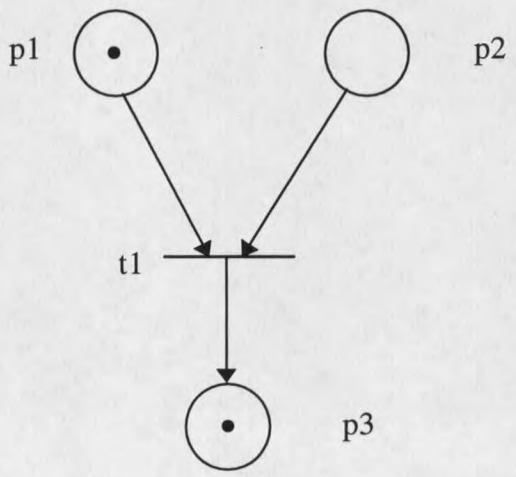

Figure 4 After transition t1 fires in Figure 3

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.

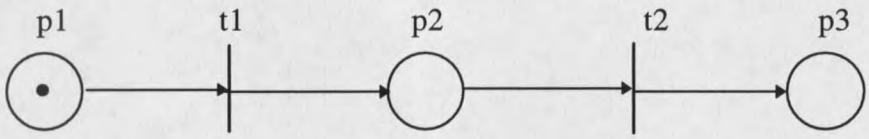

Figure 5. Sequential Execution.

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.

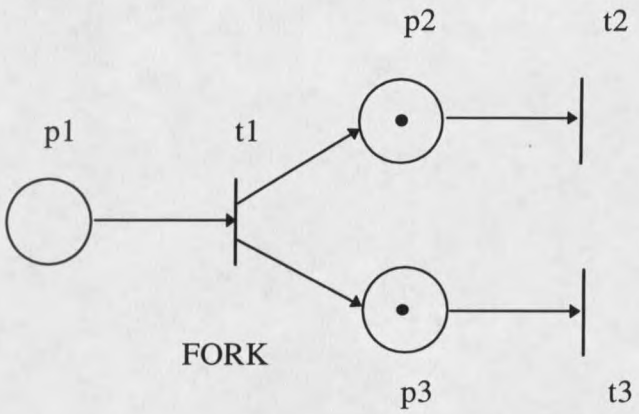

Figure 6. Concurrency.

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.

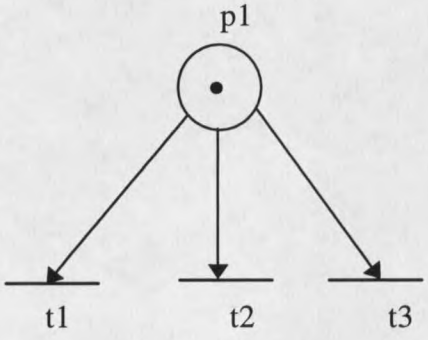

Figure 7. Conflict.

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.