Modeling Guide for Multiprocessing Systems

Matthew Bennett
School of Computing, University of Southern Mississippi
matthew.bennett@usm.edu Typeset in LaTeXon May 8, 2006

Abstract

A tremendous variety of techniques exist for the modeling and subsequent simulation of multiprocessor systems. This paper presents several of the approaches of Marsan et al. [1], in a condensed format suitable to practical systems engineer or computer scientist. Most of these techniques are based upon Generalized Stochastic Petri Nets. The guide covers Bucci's Preemptive Time Nets [2] to fill in gaps where continuous-time models fail.

Preliminaries

A working knowledge of multiprocessor architecture, statistical distributions, and the modeling technique of General Stochastic Petri Nets is required. For the latter, Murata [3] provides a succinct and quick introduction. Markov chains are mentioned, but the results presented do not require thorough understanding of the device, since the work has already been completed and proven by others.

Many of the graphs or numerical results in Marsan [1], and [3] are similar enough to be asymptotically bound for some constant. The purpose of this guide is to provide a straightforward and efficient means of guiding the systems architect through the modeling process and on into either simulation or analysis. The reader may check those sources cited for more in-depth coverage of very detailed models, since only the simplest models are presented within.

A few taxonomic niceties will save the designer some time. The modeling phase has been split into five categories (as in [1]). After the model is developed, the developer may move on to modeling fault tolerance (5) or verification (6).

1 Bus-Contention Free Architectures

The most trivial multiprocessor system to model is one that experiences no contention whatsoever. The canonical example in academia (as in [1]) is a crossbar-connected switch between n processor elements and m memory elements. An analog of crossbar switch is the Plain Old Telephone System (POTS) exchange. These are rarely seen in practice, since the number of switches and interconnections increases by a factor of $\Theta(mn)$, making the interconnection network extremely expensive for even small numbers of processors and memories.

Marsan [1] mentions that most time spent in any multiprocessing time is idle time due to contention for a common resource such as a bus. It is therefore no surprise that modeling contention-free systems is strait-forward in terms of waiting. The analysis instead concentrates on statistical modeling of random activity within the system. This is easily accomplished using queueing networks, where processors are nodes without queues, and memory modules are nodes with finite queues [1]. For a crossbar architecture, there is an edge from every processor element to every memory element, since any processor may read any memory element regardless of other processors. Some simplification occurs when two or more processors try to access the same memory node in a given time step: only one is served. Simulation can be performed using the Monte Carlo method, with memory accesses being produced using a random variable ("Markovian process" is exponential) on all producers (processors) and consumers (memories) in the net.

As more becomes known about the system under analysis, more can be included in the modeling queueing network. Marsan [1] provides a useful ontology by asking if the devices are synchronous, or asynchronous, and also whether memory accesses are uniform or non-uniform between processors.

Marsan gives a number of case studies. The simplest one is attributed to Bhandarkar [4], who assumes that memory accesses are uniform across processors, with lost requests in the case of memory contention. Bandarkar creates a vector of states that the queueing network can take on, and then further simplifies the amount of information at hand by creating super-states, or *equivalence classes* of mutually indistinguishable states (since processors and memories are assumed identical). He derives a formula (Marsan fig 6.2) which can be used to algorithmically calculate super-state transition probabilities in a Markov process, and Marsan (p. 124, 136) gives further approximation formulas from others' work which can be calculated in better time on a computer. These approximations are not as important as they once were, because computers are substantially more capable today than they were in 1986, at the time of publication. They are all valid, as they are equivalent to the exact result obtained by Bandarkar [4]. A good approximation closed-form for the number of busy memory modules β is given by Rau [5] , using binomial approximations for all events (equation 1). In this formula, m counts memories, and p counts processors.

$$\frac{\sum_{i=0}^{\min(m,p)-1} 2^i \binom{m-1}{i}\binom{p-1}{i}}{\sum_{i=0}^{\min(m,p)-1} \frac{2^i}{i+1}\binom{m-1}{i}\binom{p-1}{i}} \tag{1}$$

Delay models, in which memory contention requests are queued instead of dropped, were also considered my multiple authors. Many of the results were similar, with memory utilization expectedly being better with the addition of queues to memory [1] (136). Rau's formula given above is an excellent minimum bound for back of the envelope calculations for networks without bus contention.

2 Shared Memory Systems

Most real systems are not free of bus contention, because of the nonlinear overhead cost of building complete connection networks. When bus contention is introduced, the location of the bus with respect to the memory elements and processors plays a vital role in the performance and contention of the system as a whole. In this section, we investigate the subset of systems for which a number of processors each have a private local memory, connected by a local bus, and must contend for access to a shared memory, which requires control of one of one or more global buses. This multiprocessor architecture is common to many multiprocessor, multi-core workstations as well as supercomputers such as the classic Cray series. Classical multi-threading problems, including deadlock prevention and avoidance, starvation, and mutual exclusion must all be taken into account for any shared memory systems. Marsan [1] lays out a number of simplifying assumptions for modeling shared memory systems. Namely, they are: no delay when capturing the bus (other than normal bus contention), CSMA/CA bus discipline, and immediate release of the bus upon completion (as with capture). Marsan uses the term “active state” to mean that a processor is not currently accessing memory, “waiting state” to mean that a processor is waiting for the bus or memory, and “access state” to mean that the processor is currently in memory.

2.1 Single-Bus Shared Memory Systems

The architecture is defined by Marsan to be a single global bus connecting a number of local buses to the shared memory resource. Each local bus serves a single processor and its private memory.

He chooses three values to take into consideration: access times, active times, and

2.1.1 Exponential Distribution

The Exponential Distribution represents . In a generalized stochastic petri net, a single exponentially distributed random event can be represented by a place connected to a timed transition, where the firing rate of that transition takes on the parameter λ of the exponential distribution, and determines the likelihood of that even firing depending on a continuous time variable [1]. An example of an Exponentially distributed random variable is likelihood of a webserver receiving a packet at a particular time.

2.1.2 Erlang-k Distribution

An Erlang-K distribution models k Exponentially distributed random events which depend upon one another. They must occur in *serial*. In the Petri Net representation, Erlang-K events should be represented by a k-chain of exponential General Stochastic Continuous-Timed Petri events [1]. An example of an Erlang distributed random variable is the number of packets to a web server at a particular instant of time.

2.1.3 Hyper-Exponential Distribution

A hyper-exponentially distributed random variable counts the number of simultaneous occurrences of an exponentially distributed random event, such as processors trying to access the same piece of memory simultaneously.

2.1.4 Queueing disciplines

Marsan [1] defines three queueing disciplines for dealing with memory access contention, and mutual exclusion assurance. Fixed priority means that the processors have some preordained priority for which can preempt one another for memory access. Process sharing is like round-robin in that processors take their turn accessing memory in the order they attempt to get in, but they are guaranteed access because of some global device, like in time sharing. First come first serve simply operated like a FIFO queue, but provides no guarantee that a process will give up the bus to allow others in.

All of the combinations of these ideas can be analyzed, and have been in several different papers. Again, see Marsan p. 157 for a full bibliography of works. A simplified queuing network model or Markov chain was successfully employed to deal with simple cases in which access times and/or active times were equally exponentially distributed, but Petri nets were used by Marsan to get results for the more complex models where distribution was hyper-exponential, Erlang, or more complicated.

In all cases, the numerical results of Marsan seem to indicate that modeling with a generalized stochastic Petri net can produce approximate simulation results fairly close to the analytical results for simpler single-bus shared memory systems. The numerical results show that the queueing discipline is not an important consideration when modeling small systems. Since Petri nets are more graphical and more intuitive than analytical methods, and because the mathematics become complicated for very complex systems, the recommended method for modeling anything complex is Generalized Stochastic Petri nets as described in the 2nd half of Marsan [1] (Chapter 7 and the beginning of Chapter 8).

2.2 Multi-Bus Shared Memory Systems

Multi-bus Shared Memory systems can be modeled with Petri nets using the same techniques as single-bus shared memory systems. Generally, the number of buses are represented as tokens in a resource pool, which is a place. Depending upon the architecture of the system, and because of Marsan's simplifying assumptions for shared memory systems, immediate transitions usually follow the global bus resource pool (capturing and releasing being instantaneous, but active, access times being exponential).

Marsan gives a full analytical treatment to many complicated multi-bus systems, but that is unnecessary since most systems can be easily intuitively modeled with Petri nets. Marsan et al. first obtains upper and lower bounds limiting the gain or loss of processing power, so any numerical results given by Petri net modeling should fall within the domain on [1] (p. 186-189). This is generally a good methodology, since most bottlenecks in this configuration are assumed to stem from global bus contention.

3 Distributed Memory Systems

Shared memory systems are not the only type of distributed system. A true distributed system uses a paradigm more like message passing. This architecture consists of one or more global buses connecting local buses. Each local bus contains a processor and at least one memory element. Unlike previous shared memory systems, there is no memory connected to the global bus(es). Instead, the memory at each local bus is used by any processor in the system, and both local and global buses must be secured at each access step. An example of this type of system is a computer network utilizing file sharing. Because of the high dynamism of such networks, analysis of any but the simplest distributed memory systems is impossible with queueing nets. Marsan [1] proposes tens of architectures that are both single global bus and multiple global bus and also incorporate distributed memory, but does not give any indication of extending this methodology to generalized distributed systems.

To model any distributed memory systems, a Petri net should be employed, and numerical results obtained. The reachability tree of the Petri net can then be used to discover some analytical artifacts of the system, such as whether it is live, as described in [3].

4 Extended Petri Nets

Bucci [2] describes an extended version of Petri nets where the continuous time model is replaced with discrete time events. A global clock of some sort . A “firing event” occurs whenever the random variable distribution coincides *as well as* the time has come for that transition to fire. The idea of this extension is that a Petri net can operate within a discrete time domain, following a global clock, while keeping the ability to induce transitions from state-to-state concurrently (ie using threads or a similar mechanism). This dramatically increases the simulation uses of GSPN models, but does not really add much to their descriptive power. Bucci’s method should be taken into account any time that a simulation must occur, especially in a simulation with lots of simultaneity.

5 Fault Tolerance Modeling

If systems include repairable components, a GSPN event (place / timed transition) can represent the break and rapair of any component in the system, just as

6 Analysis and Verification

Generally, any sort of Petri Net model is good for modeling a specific system, but can only provide numerical results. In order to assure that those results are statistically significant, many runs (at least 30) should be performed in simulation, and every variable must be checked for consistency with a real system. One technique which is liberally applied by many [3] [1] is to use a verifiable Markovian process or queueing network to provide upper and lower analytical bounds. At this point, the system can simply be built and run. Analysis can be extremely difficult, so most of the time the Petri net approach will be more useful for the budding system developer, without rigorous mathematical background.

References

[1] Marsan, N. Balbo, G. Conte, G. ”Stochastic Petri Nets.” Performance Models of Multiprocessor Systems. Cambridge: MIT Press. 1986. pp. 72 - 98.

[2] Bucci, G. Correctness Verification and Performance Analysis Using Stochastic Preemptive Time Petri Nets. IEEE Transactions on Software Engineering. Vol 23, No 11. November 2005. pp. 913 - 937.

[3] Murata, T. Petri Nets: Properties, Analysis, and Applications. Invited Paper. Proceedings of the IEEE. Vol 77. No 4. April 1989. pp. 541 - 579.

[4] Bhandarkar, D. Analysis of Memory Interference in Multiprocessors. IEEE Transactions on Computers Vol 24. No 9. September 1975. 897-908.

[5] Rau, B. Interleaved Memory Bandwidth in a Model of a Multiprocessor Computer System. IEEE Transactions on Computers Vol 28. No 9. September 1979. 678 - 681.