

Multiple Processor Systems for Real-Time Applications

October 2007

Multiple Processor Systems for Real-Time Applications, by Burt Liebowitz and John Carson,¹ is a classic treatise on distributed systems. Written in the time when 64k was a massive amount of memory, it predicted tiered systems and clusters.

Not only is this book a joy to read to see how early technology morphed into today's powerful distributed systems, but more importantly it provides in-depth discussions of reliability theory and performance analysis with a focus on distributed systems. The material presented in these chapters is every bit as pertinent today as it was then and is a must-read for any serious distributed system architect.

Background

Liebowitz and Carson were each deeply involved in distributed systems, both in theory and in practice. They were responsible for the implementation of many early distributed systems. They jointly and individually taught a series of courses at George Washington University in Washington, D.C. as well as to government and industry groups on the subject of distributed systems.

This book evolved from the extensive course notes supporting these classes.

Scope

This book deals specifically with locally distributed multiple processor systems (MPS). An MPS is a network of independent computers cooperating in an application. The authors distinguish MPS from multiprocessor systems, which are standard symmetric multiprocessors (SMP) in which multiple CPUs share the same memory.

A locally distributed MPS is one in which the computers are close enough (in the same room, building, or campus) so that high-speed communication links such as a LAN can be used as the interconnect facility. The book discusses the design issues associated with locally distributed MPS systems as well as the consideration for the choices of components. Though this material is quite dated, much of it still holds true today.

A particularly complete discussion of reliability theory and performance analysis focuses on distributed systems. Both of these discussions cover serial tiers of redundant processing elements that are the heart of distributed systems. Estimating the availability and response time of these systems can be a challenge, and these chapters present the tools required to meet this challenge.

The book ends with a series of in-depth case studies of the period and some predictions for the future – predictions that in hindsight hit the mark.

¹ Liebowitz, Burt H.; Carson, J. H.; *Multiple Processor Systems for Real-Time Applications*, Prentice-Hall; 1985.

Advantages of Multiple Processor Systems

The authors point out that multiple processor systems have several advantages over multiprocessor systems and monolithic mainframes.

- *Reliability* – An MPS can include incremental redundancy to protect the system against any single point of failure. Though they incorporate multiple CPUs and can withstand a CPU failure, multiprocessor systems depend upon a common memory that is a single point of failure. Mainframes are by their very nature essentially nonredundant.
- *Scalability* – An MPS can be easily scaled by adding computers to the network. A multiprocessor can add CPUs but only to a point. At some point, the memory becomes a bottleneck. Once the capacity of a mainframe is reached, the only option is to go to a bigger box.
- *Performance* – The performance advantages of an MPS parallel its scalability. If a certain class of transactions begins to take too long because of loading on a component such as a CPU or a disk, there is the opportunity to add additional components to share the load. In addition, the MPS can be structured in advance with load and performance requirements in mind.
- *Program Modularity* – As opposed to mainframes and multiprocessors, the programs in an MPS can be highly modularized by distributing functions across multiple computers. This leads to easier development and simpler testing.
- *Economy* – A properly structured MPS can be an order of magnitude less expensive than a mainframe.
- *Security* – Redundant elements of an MPS can be placed in different locations to protect it against disasters such as a fire.

Design Issues

The design of an MPS goes far beyond that required for a monolithic mainframe or multiprocessor system. In effect, the designer becomes a system architect. He must

- decompose the problem into a manageable number of computers.
- minimize interprocessor communication and its overhead.
- define a testable collection of processors.
- provide a starting point for resolving other architectural issues.
- provide a modular, expandable system.
- provide a configuration that is amenable to incremental redundancy.

There are several configurations that the authors consider for an MPS.

- *Dedicated Function* – each computer in the network is dedicated to a particular function, such as communications, different applications, and database management. This is the basis for the tiered system architectures of today.
- *Traffic Sharing* – traffic is distributed to an array of two or more computers that, in the aggregate, are capable of supporting the entire load. This is the load-sharing architecture of today so prevalent in web systems. It is inherently fault-tolerant in that, should a processing component fail, it is simply not fed any new traffic.

- *Dynamically Allocated* – Functions are assigned to computers in the network on an as-needed basis via a control mechanism that optimizes component utilization. This architecture is inherently reliable since, should a computer fail, it is simply not assigned any new tasks until it is returned to service. This is similar to today's cluster technology.

In these architectures, the computers may be organized in either a hierarchical or a horizontal relationship. In a hierarchical relationship, one computer serves as a master to the other subordinate computers. The computers are logically unequal. The master computer controls the flow of work to its subordinates.

In a horizontal relationship, the computers are connected by a bus and are logically equal. Control is distributed.

The Building Blocks

An MPS is constructed from a variety of building blocks. These include the processors, database systems, interconnect systems, and operating systems.

The book discusses the considerations that go into the choices of these components. Though much of this material is dated and is primarily of historic interest, some topics are as applicable today as they were then.

The OSI model is explained in detail, and the concept of flow control and windowing is clearly demonstrated.

The concept of token networks and broadcast (CSMA/CD) buses is treated in detail. It is shown that the capacity utilization of the LAN at low usage is poor for a token bus but is high for a broadcast bus. However, as utilization goes up, the efficiency of a token bus increases dramatically while the efficiency of a broadcast bus deteriorates quickly (due to collisions). The efficiency of a token ring remains high regardless of the LAN load.

Various methods for distributing a database are described. These include distributing partitions, replication, and functional distribution.

The book notes that the requirements for a distributed operating system are substantially more complex than those for a monolithic operating system. The Tandem (now HP NonStop) Guardian operating system is used as an example.

Recovery

An extensive description of failover considerations is given by the authors. These considerations are every bit as applicable today and include:

- How is a faulty component identified?
- Who makes the decision to switch to a new component?
- How does a new computer acquire the peripherals of a failed computer?
- How can the database be preserved or reconstructed?
- What mechanism should be used for loading programs into the new computer?
- How much should be done automatically, and how much should be done under operator control?
- How many spares are required to achieve the required reliability?
- What failure modes can be introduced by the diagnostic and switching equipment?

These are organized into three categories – error detection, fault isolation, and recovery – for further discussion.

These considerations are applied to a variety of MPS architectures.

Reliability

The book's discussion on reliability analysis is one of the enduring sections that is still relevant today. The authors distinguish between availability and reliability:

- *Availability* is the proportion of time that the system is operational (*interval availability*) or, alternatively, the probability that the system will be operational at a specific point in time, like when you walk up to an ATM (*pointwise availability*). In the steady state, these are the same. If a system has an availability of 99%, it will be operational 99% of the time and will be there when you need it 99% of the time.
- *Reliability* is the probability that the system will be operational for a stated period of time. It is characterized by a statistical distribution. For instance, a particular system might have a 90% probability that it will be operational for a year, a 30% probability that it will be operational for five years, and a 5% probability that it will be operational for 10 years.

If a system is repairable, availability is usually used as a measure of the operating performance of the system. However, if the system is not repairable, availability has no meaning. In this case, reliability is the more important measure. A good example of a nonrepairable system is a satellite subsystem. In this case, the measure of interest is how long the system is expected to be operational in orbit.

Availability

The authors first describe the techniques for analyzing networks of components. Each component has an availability that is determined by its mean time before failure, mtbf, and its mean time to repair, mtr. Using these values, component availability is

$$\text{availability} = \text{mtbf}/(\text{mtbf}+\text{mtr})$$

These networks comprise two types of subnetworks. One is a serial arrangement in which every component must work in order for the system to be operational. The other is a parallel arrangement in which the system will work even if one or more components should fail. Any system can be decomposed into serial and parallel components. The techniques for analyzing the availability of such a system are described in detail.

Parallel configurations of redundant components are further analyzed. The availability relationship for an n-node system with s spares is derived and is applied to a multiple computer configuration.

Reliability

The authors then turn their attention to the calculation of reliability. Using the same component measurements of mtbf and mtr, they show the method for calculating the mtbf and mtr for the system as a whole. This is what is needed for the expression for reliability. Their method depends upon S. J. Einhorn's work, which is equivalent to the expressions that we have developed in our previous [Calculating Availability](#) articles.

System mtbf calculated in this manner is the average time that the system is expected to remain operational. It is the time that the system will be operational with a probability of 50%. As such, it is just one point on the reliability curve; but it is the parameter generally used to characterize the reliability of a system.

Again, these expressions are applied to a multinode MPS system with multiple spares as an example.

Performance

The authors present an equally impressive coverage of response time for MPS systems. They first review basic queuing theory for the single server model for service times with arbitrary distributions, resulting in the well-known Pollaczek-Khinchine model. They then solve this model for service times with random distributions and with constant distributions.

They then extend this to response time distributions so that questions such as “what is the response time for 90% of the requests?” can be answered.

These results are further extended to the complex topic of multiserver queues in which multiple servers process a common request stream in a load-sharing configuration. This is the heart of many distributed systems.

Tandem queues and parallel queues are also analyzed.

Finally, this analysis is applied to a fairly complex redundant system, clearly illustrating the use of these methods. The system comprises up to sixteen processors, including dual I/O processors (one acting as a standby spare), dual communication line controllers, dual disk systems, and multiple application processors. This is perhaps one of the most complete and complex performance analyses that I personally have ever seen.

Design Methodology

The authors discuss a systematic approach to resolving the many issues that arise during the architectural phase. The discussion begins with the importance of a full functional requirements specification – a document often missing or inadequate at the start of a project.

The first step in this design methodology is to size the application on a “standard” computer. This may be a known computer or one with defined characteristics. The goal here is to estimate the CPU and disk loads required by the application.

Given these load requirements, a distributed system architecture with appropriate redundancy can be determined and computing nodes chosen. The choice of architecture and components should be based on simplicity, feasibility, modularity, maintainability, expandability, reliability, cost, and performance.

At this point, implementation proceeds as it would for a monolithic system.

This design approach is then applied as an extensive example to a store-and-forward packet switch.

Case Histories

Several actual case histories of MPS systems are described. They include systems implemented by the Coast Guard, the FAA, Bank of America, NASDAQ, and E. F. Hutton.

Future Trends

The authors predict some future trends, all of which have come to pass:

- More users will demand online, real-time access to data.

- Reliability will become a concern as applications go online.
- Users will require systems that expand gracefully with increasing traffic.
- The cost/performance characteristic for small computers will continue to improve.
- Users will demand low, predictable response times.
- More and more off-the-shelf products will become available to facilitate the construction of distributed systems.

Sound familiar?

The book ends with detailed descriptions of some packaged multiple processor systems, including those from Tandem (now HP NonStop), Stratus, and Synapse (long since gone).

Summary

From a historical perspective, it is interesting to note the rapid advance of technology. System characteristics have improved by three to six orders of magnitude (that's one million times) over twenty-some-odd years! Memory sizes have gone from kilobytes to gigabytes. Disk capacity has gone from megabytes to terabytes. Processor speeds have gone from megahertz to gigahertz. WAN communication speeds have gone from kilobits per second to megabits per second.

As systems become more powerful and complex, we have gotten further and further away from the internals of our systems. In reading this book, it is amazing how close we were to system internals back then.

Historical perspective aside, the authors' treatment of reliability and performance analysis is complete and concise and is as relevant and helpful as it ever has been.

The authors summarize their message succinctly:

"There is no cookbook for the design process. One must gather detailed information on both the application and the equipment selected in order to proceed with a design. What we really stress is the need for quantitative analysis during the selection and design process. Whether it be simple addition to determine the total memory and disk requirements or more complex mathematics, such as queuing or reliability analysis, we must perform this analysis if we are to do better than make a blind guess at a solution. We are repeatedly amazed at how many systems are attempted without any sincere analytical study preceding the design. We are not amazed, however, at how many systems of this type never perform as anticipated."

This is a timeless statement.