

Distributed and Multiprocessor Scheduling

Steve J. Chapin, Kent State University

Introduction

We discuss CPU scheduling in parallel and distributed systems. CPU scheduling is part of a broader class of resource allocation problems, and is probably the most carefully studied such problem. The scheduling problem for multiprocessor systems can be generally stated as “How can we execute a set of tasks T on a set of processors P subject to some set of optimizing criteria C ?” The most common goal of scheduling is to minimize the expected runtime of a task set. Examples of other scheduling criteria include minimizing the cost, minimizing communication delay, giving priority to certain users’ processes, or needs for specialized hardware devices. The scheduling policy for a multiprocessor system usually embodies a mixture of several of these criteria.

Issues in Multiprocessor Scheduling

Solutions to the scheduling problem come in two general forms: algorithms and scheduling systems. Algorithms concentrate on policy while scheduling systems provide mechanism to implement the algorithms. Some scheduling systems run outside the operating system kernel, while others are part of a tightly-integrated distributed or parallel operating system. Distributed systems communicate via message-passing, while parallel systems use shared memory.

A task is the unit of computation in computing systems, and a job consists of one or more cooperating tasks. Global scheduling involves assigning a task to a particular processor within the system. Local scheduling determines which of the set of available tasks at a processor runs next on that processor. Task migration can change the global mapping by moving a task to a new processor. If we have several jobs, each composed of many tasks, we can either assign several processors to a single job, or we can assign several tasks to a single processor. The former is known as space sharing, and the latter is called time sharing.

Global scheduling is often used to perform load sharing. Load sharing allows busy processors to offload some of their work to less busy processors. Load balancing is a special case of load sharing, in which the goal is to keep the load even across all processors. Sender-initiated load sharing occurs when busy processors try to find idle processors to offload some work. Receiver-initiated load sharing occurs when idle processors seek busy processors. It is now accepted wisdom that full load balancing is generally not worth doing, as the small gain in execution time over simpler load sharing is more than offset by the effort expended in maintaining the balanced load.

As the system runs, new tasks arrive while old tasks complete execution (or are *served*). If the arrival rate is greater than the service rate then the system is said to be unstable. If tasks are serviced as least as fast as they arrive, the system is said to be stable. If the arrival rate is just slightly less than the service rate for a system, an unstable scheduling policy can push the system into instability. A stable policy will never make a stable system unstable.

Distributed Scheduling

In most cases, work in distributed scheduling concentrates on global scheduling because of the architecture of the underlying system. Casavant and Kuhl [CK88] defines a taxonomy of scheduling algorithms for distributed systems. The two major categories of global algorithms are static and dynamic.

Static algorithms make scheduling decisions based purely on information available at compilation time, such as the number of processors in the system and the number of tasks. Dynamic algorithms take the current system state as input to the scheduling policy. Adaptive algorithms are a special subclass of dynamic algorithms, and are often discussed separately. Adaptive algorithms go one step further than dynamic algorithms, in that they may completely switch policy based on dynamic information.

The list of scheduling algorithms that appears in the literature is too long to reproduce here. Instead, readers are referred to the following references for detailed examination of these algorithms. A representative sample of these algorithms are described in [CK88, Cha93, SHK95, SS94].

There are several algorithms that perform static analysis on a task set and generate a static task mapping for a particular architecture. Each of these tools represents the task set as a directed acyclic graph. These tools attempt to map the static task graph onto a given machine. Algorithms that perform static analysis include Parallax, Hypertool, macro-dataflow, Prep-P, and Oregami [SHK95].

Shared-Memory Parallel Systems

Researchers working on shared-memory parallel systems have concentrated on local scheduling because of the ability to trivially move processes between processors. There are two main causes of delay that can be introduced by local scheduling in these systems: cache corruption caused by moving processes between processors and wasted cycles due to pre-emption of processes holding locks. Techniques to avoid these problems are coscheduling, Smart Scheduling, and affinity-based scheduling [SS94].

Distributed-Memory Systems

In distributed-memory systems, such as hypercube systems, there is global scheduling done. Most hypercube systems use space sharing, in that they reserve subcubes of the larger hypercube for use by a single application.

The PUMA [WMR⁺94] operating system running on the Intel Paragon message-passing supercomputer combines space sharing and time sharing. Users can reserve blocks of processors for their use (space sharing), and can run multiple processes on each processor (time sharing).

Distributed Scheduling Support Systems

Scheduling support mechanisms focus more on supplying the ability to perform distributed scheduling, rather than concentrating on specific algorithms for making scheduling decisions.

Several systems have been built to support scheduling in locally-distributed systems, including Condor and Stealth [SS94].

MESSIAHS [Cha93] is intended for use in wide-area distributed systems comprised of locally-distributed systems that may be running scheduling software such as Condor. Legion [GWF⁺94], SmartNet [HMK⁺95], and Utopia [ZZWD93] also provide the capability to do distributed scheduling in large-scale distributed systems.

The Schizophrenic Workstation system, or Schizo, is a distributed operating system personality that is built on top of Mach [SS94]. As a distributed operating system, it has tighter control over the component systems than is exerted by the scheduling systems mentioned above. Other distributed operating systems include Sprite, the V System, Locus, and MOSIX [SS94].

Research Issues and Summary

The central problem in distributed scheduling is assigning a set of tasks to a set of processors in accordance with one or more optimizing criteria. We have reviewed many of the algorithms and mechanisms developed thus far to solve this problem. However, much work remains to be done. The primary areas of new research are heterogeneity and autonomy. Accommodating heterogeneity is primarily a technical problem; autonomy, on the other hand, is primarily a social problem. We must develop scheduling systems that support the autonomy of local systems if we expect the owners of those systems to make them available for at-large computation in a large-scale distributed system.

For Further Information

Many of the seminal theoretical papers in the area are contained in *Scheduling and Load Balancing in Parallel and Distributed Systems*, edited by Shirazi, Hurson, and Kavi [SHK95]. Chapin [Cha93] contains a survey of over 40 scheduling algorithms. Singhal and Shivaratri [SS94] describes and gives pointers to many well-known distributed scheduling systems and distributed operating systems. These volumes serve as excellent starting points for those interested in further reading in the area.

References

- [Cha93] S. J. Chapin. Scheduling Support Mechanisms for Autonomous, Heterogeneous, Distributed Systems. Ph.D. Dissertation, Purdue University (Purdue CS TR-93-087), 1993.
- [CK88] T. L. Casavant and J. G. Kuhl. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, February 1988.
- [GWF⁺94] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds Jr. Legion: The next logical step toward a nationwide virtual computer. Technical Report CS-94-21, Dept. of Computer Science, University of Virginia, June 1994.
- [HMK⁺95] D. Hensgen, L. Moore, T. Kidd, R. Freund, E. Keith, M. Kussow, J. Lima, , and M. Campbell. Adding rescheduling to and integrating condor with smartnet. In

- Proceedings of the 4th Heterogeneous Computing Workshop*, pages 4–11. IEEE, 1995.
- [SHK95] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, 1995. ISBN 0-8186-6587-4.
- [SS94] M. Singhal and N. G. Shivaratri. *Advanced Concepts in Operating Systems*. McGraw–Hill, 1994. ISBN 0-07-13668-1.
- [WMR⁺94] S. R. Wheat, A. B. Maccabe, R. Riesen, D. W. van Dresser, and T. M. Stallcup. PUMA: An operating system for massively parallel systems. *Scientific Programming*, 3:275–288, 1994. special issue on Operating System Support for Massively Parallel Computer Architectures.
- [ZZWD93] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software—Practice and Experience*, 23(12):1305–1336, 1993.