A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing

(Extended Abstract)

Derek L. Eager

Department of Computational Science
University of Saskatchewan

Edward D. Lazowska and John Zahorjan

Department of Computer Science
University of Washington

One goal of locally distributed systems is to facilitate resource sharing. Most current locally distributed systems, however, share primarily data, data storage devices, and output devices; there is little sharing of *computational* resources. *Load sharing* is the process of sharing computational resources by transparently distributing the system workload. System performance can be improved by transferring work from nodes that are heavily loaded to nodes that are lightly loaded.

Load sharing policies may be either *static* or *adaptive*. Static policies use only information about the average behavior of the system; transfer decisions are independent of the actual current system state. Static policies may be either *deterministic* (e.g., "transfer all compilations originating at node A to server B") or *probabilistic* (e.g., "transfer half of the compilations originating at node A to server B, and process the other half locally").

Numerous static load sharing policies have been proposed. Early studies considered deterministic rules [Stone 1977, 1978; Bokhari 1979]. More recently, Tantawi and Towsley [1985] have developed a technique to find optimal probabilistic rules.

The principal advantage of static policies is their simplicity: there is no need to maintain and process system state information. Adaptive policies, by contrast, are more complex, since they employ information on the current system state in making transfer decisions. This information makes possible significantly greater performance benefits than can be achieved under static policies. This potential was clearly indicated by Livny and Melman [1982], who showed that in a network of homogeneous, autonomous nodes there is a high probability that at least one node is idle while tasks are queued at some other node, over a wide range of network sizes and average node utilizations.

This material is based upon work supported by the National Science Foundation under Grants MCS-8004111, MCS-8302383, and DCR-8352098, and by the Natural Sciences and Engineering Research Council of Canada. This work was conducted while Lazowska was on leave at Digital Equipment Corporation's Systems Research Center.

Authors' Addresses: Derek L. Eager, Department of Computational Science, University of Saskatchewan, Saskatoon, Canada, S7N 0W0; Edward D. Lazowska and John Zahorjan, Department of Computer Science FR-35, University of Washington, Seattle, WA 98195.

In previous work [Eager, Lazowska & Zahorjan 1984] we considered the appropriate level of complexity for adaptive load sharing policies. (For example, how much system state information should be collected, and how should it be used in making transfer decisions?) Rather than advocating specific policies, we considered fairly abstract strategies exhibiting various levels of complexity. We demonstrated that the potential of adaptive load sharing can in fact be realized by quite simple strategies that use only small amounts of system state information. This result is important because of a number of practical concerns regarding complex policies: the effect of the overhead required to administer a complex policy, the effect of the inevitable inaccuracies in detailed information about system state and workload characteristics, and the potential for instability. (We consciously use the phrase "load *sharing*" rather than the more common "load *balancing*" to highlight the fact that load *balancing*, with its implication of attempting to equalize queue lengths system-wide, is not an appropriate objective.)

Adaptive load sharing policies can employ either *centralized* or *distributed* control. Distributed control strategies can be of two basic types (although intermediate strategies also are conceivable): *sender-initiated* (in which congested nodes search for lightly loaded nodes to which work may be transferred), and *receiver-initiated* (in which lightly loaded nodes search for congested nodes from which work may be transferred). Our earlier paper considered distributed, sender-initiated policies – a sufficiently rich class to allow us to answer the fundamental questions of policy complexity that we were addressing. In the course of understanding the reasons for the degradation of these policies at high system loads, we were led to consider receiver-initiated policies as a possible alternative. The comparison of receiver-initiated and sender-initiated adaptive load sharing is the purpose of the present paper.

There have been several experimental studies, using prototypes and simulation models, of specific (typically fairly complex) adaptive load sharing policies [Bryant & Finkel 1981; Livny & Melman 1982; Kreuger & Finkel 1984; Barak & Shiloh 1984]. Both sender-initiated policies and receiver-initiated policies have been considered. However, there has not previously been a rigorous comparison of these two strategies. Such a comparison is made difficult by the problem of choosing appropriate representative policies of each type, and by the potentially quite different costs incurred in effecting transfers. (Receiver-initiated policies typically will require the transfer of executing tasks, which incurs substantial costs in most systems [Powell & Miller 1983]. Sender-initiated policies naturally avoid such costly transfers, since tasks can be transferred upon arrival, prior to beginning execution.)

Our present paper is similar to our previous work in that our purpose, rather than to advocate specific policies, is to address a fundamental question concerning policies in general: How should system state information be collected and load sharing actions initiated – by potential *receivers* of work, or by potential *senders* of work? In studying this question we consider a set of abstract policies that represent only the essential aspects of receiver-initiated and

sender-initiated load sharing strategies. These policies are investigated using simple analytic models. Our objective is not to determine the absolute performance of particular load sharing policies, but rather to gain intuition regarding the relative merits of the different approaches under consideration.

We represent locally distributed systems as collections of identical nodes, each consisting of a single processor. The nodes are connected by a local area network (e.g., an Ethernet). All nodes are subjected to the same average arrival rate of tasks, which are of a single type.

In contrast to most previous papers on load sharing, we represent the cost of task transfer as a processor cost rather than as a communication network cost. It is clear from measurement and analysis [Lazowska et al. 1984] that the processor costs of packaging data for transmission and unpackaging it upon reception far outweigh the communication network costs of transmitting the data.

We study three abstract load sharing policies, comparing their performance to each other and to that of a system in which there is no load sharing. The *Sender* policy is used as a representative of sender-initiated load sharing strategies. The *Receiver* and *Reservation* policies are used as representatives of receiver-initiated load sharing strategies; unlike the Receiver policy, the Reservation policy will transfer only newly arriving tasks. In a bit more detail:

Sender

In our earlier work concerning the appropriate level of complexity for adaptive load sharing schemes, we identified two sub-policies of sender-initiated strategies. The ***transfer policy*** determines whether a task should be processed locally or remotely. The ***location policy*** determines to which node a task selected for transfer should be sent.

In that previous study, we considered ***threshold*** transfer policies, in which each node uses only local state information. An attempt is made to transfer a task originating at a node if and only if the number of tasks already in service or waiting for service (the node ***queue length***) is greater than or equal to some threshold T.

We considered various location policies spanning a range of complexity. We found that the use of a complex location policy yields only slight improvement over the use of a simple location policy that, like the transfer policy, uses threshold information. In this ***threshold*** location policy, a node is selected at random and ***probed*** to determine whether the transfer of a task to that node would place the node above the threshold T. If not, then the task is transferred. If so, then another node is selected at random and probed in the same manner. This continues until either a suitable destination node is found, or the number of probes reaches a static ***probe limit***, L_p. In the latter case, the originating node must process the task. (The use of probing with a fixed limit, rather than broadcast, ensures that the cost of executing the load sharing policy will not be prohibitive even in large networks. The performance of this policy was found to be surprisingly insensitive to the choice of probe limit: the performance with a small probe limit, e.g., 3 or 5, is nearly as good as the performance with a large probe limit, e.g., 20.)

The sender-initiated policy with a threshold transfer policy and a threshold location policy was found to yield performance not far from optimal, particularly at light to moderate system loads. For this reason, and because of its simplicity, we choose this policy to serve as the representative of sender-initiated strategies for the comparison that is the subject of the present paper, and term it here the Sender policy.

Receiver

To facilitate comparison between sender-initiated strategies and receiver-initiated strategies, a representative policy of the latter class should be as similar as possible to the Sender policy. In particular, it should utilize threshold-type state information, and have a bound L_p on the number of remote nodes whose state can be examined when making a task transfer decision.

In the Receiver policy, a node attempts to replace a task that has completed processing if there are less than T tasks remaining at the node. A remote node is selected at random and probed to determine whether the transfer of a task from that node would place its queue length below the threshold value T. If not, and if the node is not already in the process of transferring a task, a task is transferred to the node initiating the probe. Otherwise, another node is selected at random and probed in the same manner. This continues until either a node is found from which a task can be obtained, or the number of probes reaches a static probe limit, L_p. In the latter case, the node must wait until another task departs before possibly attempting again to initiate a transfer. (This is completely analogous to the operation of the Sender policy, in which a node that fails to find a suitable destination to which to transfer a task must wait until another task arrives before attempting again to initiate a transfer.) The Receiver policy with $T=1$ has been studied using a simulation model by Livny and Melman [1982], who term it the "poll when idle algorithm".

Reservation

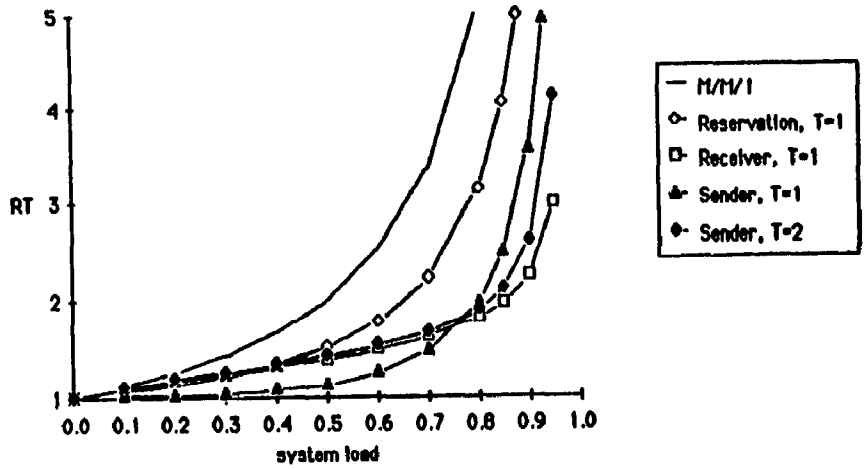
The Reservation policy, like the Sender policy but in contrast to the Receiver policy, will only transfer newly arriving tasks. This may be advantageous in multiprogramming systems in which nodes attempt to give each of the tasks present some share of the total available processing power. If the Receiver policy is used in such a system, almost all task transfers will involve executing tasks, and may be substantially more costly than transfers of non-executing tasks.

In the Reservation policy, as in the Receiver policy, a node attempts to replace a task that has completed processing if there are less than T tasks remaining at the node. A remote node is selected at random and probed to determine whether the transfer of the next task to originate at that node would place its queue length below the threshold value T. If not, and if no other "reservation" is pending for this node, then this next arrival is "reserved" by the probing node; it is transferred upon arrival if no other tasks have arrived at the probing node by that time. If the reservation attempt is not successful, another node is selected at random and probed in the same manner. This continues until either a node is found at which the next arrival can be reserved, or the number of probes reaches a static probe limit, L_p. In the latter case, the node must wait until another task departs before possibly attempting again to reserve a task.

Our evaluation of this policy is optimistic. (Even this optimistic evaluation predicts unsatisfactory performance.) At the time a reservation is attempted, we assume that the probed node can "see into the future" to the arrival time of the (potentially) reserved task. The reservation is made only if the probed node will be above threshold at that time. Also, even when a reservation request is successful, the probed node considers this next arrival as ineligible for other reservation requests only if it will actually be transferred to the node holding the reservation. Finally, we assume that the probability that a task will be processed locally rather than transferred, given that it arrives when the node queue length is at or over threshold, is independent of the prior history of task arrivals and departures. In fact, this probability is higher for tasks with shorter interarrival times.

Many of the results of our study are illustrated in the accompanying figure. While the figure illustrates specific choices of parameter values, the results are quite robust with respect to these choices; a substantial part of the full paper is devoted to demonstrating this robustness. The results include:

- Both receiver-initiated and sender-initiated policies offer substantial performance advantages over the situation in which no load sharing is attempted (shown as M/M/1 in the figure).
- Sender-initiated policies are preferable to receiver-initiated policies at light to moderate system loads.
- Receiver-initiated policies are preferable at high system loads, but only if the costs of task transfer under the two strategies are comparable.



S (task service time) $= 1$ C (cost of task transfer) $= 0.1$ L_p (probe limit) $= 3$

Principal Performance Comparison: Response Time vs. Load ρ

- If the cost of task transfers under receiver-initiated policies is significantly greater than under sender-initiated policies (for example, because executing tasks must be transferred), then sender-initiated policies provide uniformly better performance.
- Modifying receiver-initiated policies to transfer only newly-arrived tasks (so as to avoid the cost of transferring executing tasks) yields unsatisfactory performance.

References

[Barak & Shiloh 1984]
A. Barak and A. Shiloh. A Distributed Load Balancing Policy for a Multicomputer. Department of Computer Science, The Hebrew University of Jerusalem, 1984.

[Bokhari 1979]
S.H. Bokhari. Dual Processor Scheduling with Dynamic Reassignment. *IEEE Transactions on Software Engineering SE-5*,4 (July 1979), pp. 341-349.

[Bryant & Finkel 1981]
R. Bryant and R.A. Finkel. A Stable Distributed Scheduling Algorithm. *Proc. 2nd International Conference on Distributed Computing Systems* (April 1981), pp. 314-323.

[Eager, Lazowska & Zahorjan 1984]
D.L. Eager, E.D. Lazowska and J. Zahorjan. Adaptive Load Sharing in Homogeneous Distributed Systems. Technical Report 84-10-01, Department of Computer Science, University of Washington, October 1984. Submitted to *IEEE Transactions on Software Engineering.*

[Lazowska et al. 1984]
E.D. Lazowska, J. Zahorjan, D.R. Cheriton and W. Zwaenepoel. File Access Performance of Diskless Workstations. Technical Report 84-06-01, Department of Computer Science, University of Washington, June 1984. To appear in *ACM Transactions on Computer Systems.*

[Livny & Melman 1982]
M. Livny and M. Melman. Load Balancing in Homogeneous Broadcast Distributed Systems. *Proc. ACM Computer Network Performance Symposium* (April 1982), pp. 47-55.

[Powell & Miller 1983]
M.L. Powell and B.P. Miller. Process Migration in DEMOS/MP. *Proc. 9th ACM Symposium on Operating Systems Principles* (October 1983), pp. 110-119.

[Stone 1977]
H.S. Stone. Multiprocessor Scheduling with the Aid of Network Flow Algorithms. *IEEE Transactions on Software Engineering SE-3*,1 (January 1977), pp. 85-93.

[Stone 1978]
H.S. Stone. Critical Load Factors in Two Processor Distributed Systems. *IEEE Transactions on Software Engineering SE-4*,3 (May 1978), pp. 254-258.

[Tantawi & Towsley 1985]
A.N. Tantawi and D. Towsley. Optimal Static Load Balancing in Distributed Computer Systems. *JACM 32*,2 (April 1985), pp. 445-465.