LOAD BALANCING USING TIME SERIES ANALYSIS FOR SOFT REAL TIME SYSTEMS WITH STATISTICALLY PERIODIC LOADS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Max Hailperin
December, 1993

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Anoop Gupta
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John L. Hennessy

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Harold Brown

Approved for the University Committee on Graduate Studies:

Abstract

This thesis provides design and analysis of techniques for global load balancing on ensemble architectures running soft-real-time object-oriented applications with statistically periodic loads. It focuses on estimating the instantaneous average load over all the processing elements.

The major contribution is the use of explicit stochastic process models for both the loading and the averaging itself. These models are exploited via statistical time-series analysis and Bayesian inference to provide improved average load estimates, and thus to facilitate global load balancing.

This thesis explains the distributed algorithms used and provides some optimality results. It also describes the algorithms' implementation and gives performance results from simulation. These results show that our techniques allow more accurate estimation of the global system loading, resulting in fewer object migrations than local methods. Our method is shown to provide superior performance, relative not only to static load-balancing schemes but also to many adaptive load-balancing methods. Results from a preliminary analysis of another system and from simulation with a synthetic load provide some evidence of more general applicability.

Acknowledgements

I would like to thank not only my advisor, Anoop Gupta, but also my other two mentors who worked closely with me on this project: Bruce Delagi and Harold Brown. It was they who provided the application and architectural context, who suggested the general research area of load balancing to me, and who encouraged me through the four years I spent working on this problem at Stanford. (In fact, Harold has egged me on in the three years since I left Stanford as well, because I still remember his parting words to me: "Don't become another ....") Of course it goes without saying that Anoop was also instrumental in this process. In particular, it was he who was most closely critical of my research methodology and more recently of my exposition. I thank him for this constructive criticism, and for the boundless patience I've given him opportunity to exhibit. Harold, Bruce, and John Hennessy also showed their share of patience, and I thank them for that as well. I'd also like to thank Ed Feigenbaum for consistently showing interest in my work throughout this period, and for making the Advanced Architectures Project happen.

Other less celebrated figures at Stanford also contributed essential help to my work. Nakul Saraiya deserves singular mention among these; he did the majority of the development of the simulator, programming environment, and application program that were central to my experimental work. He also cheerfully allowed me to pester him constantly for help and advice when I had technical difficulties. Rich Acuff performed lisp machine miracles on a routine basis. Greg Byrd, Sayuri Nishimura, Alan Noble, and James Rice also were most helpful. I'm most grateful to all these people for their technical help, as well as their camaraderie. Greg Byrd, my long term cubicle-mate, deserves special thanks in the camaraderie department.

Three years ago, I left Stanford with my thesis not quite done so that I could teach at Gustavus Adolphus College. While three years may seem like a long time for the transition from not quite done to done, I am acutely aware that that transition would never have occurred at all had it not been for the absolutely incredible amount of support I've received from my colleagues at Gustavus, particularly Karl Knight and Barbara Kaiser. They have been by turns sympathetic, protective, encouraging, and dictatorial, as necessary to get me off my duff. Karl has patiently read drafts of most of the chapters, talked with me extensively about them, and given extremely useful suggestions for improvements. I won't try to enumerate my entire 13-member department, but others have been quite encouraging as well. Thanks very much to all my friends here at Gustavus.

Last but not least, thanks to my wife Stefanie and my parents, who told me that they "knew" I'd finish one day, even when I didn't know that, and then proceeded to tell me it would be o.k. with them if I didn't. My son Karl, who doesn't know that he's been helping me through a thesis (because he doesn't know what one is) has none the less done so, by brightening up even my grimmest days.

This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation. This work was also supported by DARPA Contracts F30602-85-C-0012 and MDA903-83-C-0335, NASA Ames Contract NCC 2-220-S1, Boeing Contract W266875, and Digital Equipment Corporation.

Contents

List of Figures

Chapter 1

Introduction

As the peak performance of computer systems has increased, one increasingly common research theme has become the difficulty of actually approximating that peak performance in practice. In particular, although concurrent computation can raise the peak performance of a system in proportion to the number of processing elements, careful load distribution is necessary to prevent imbalances from detracting substantially from the peak performance.

Although this theme is quite general, the specific challenges vary with hardware and software architecture and with application domain. Less obviously, the opportunities for solutions vary with architecture and application as well. This thesis addresses a specific combination of architectural and application types that provides a particularly interesting mix of challenges and opportunities. Rather than attempting to find a general "silver bullet" for all load distribution problems, we will examine carefully how to fully exploit the particular opportunities afforded by our chosen domain so as to meet its specific challenges.

Concretely, this thesis examines a class of soft-real-time surveillance applications implemented in an object-oriented style on an ensemble multicomputer. We show that this domain presents the load distribution challenge of performing global load balancing without rapid global information dissemination. However, we also show that this domain can provide a unique opportunity, namely statistically periodic loading, and show how that opportunity can be exploited in order to meet the global load balancing challenge. The following sections of this introduction outline in more detail the challenge, the opportunity, and the contributions of the thesis, concluding with a brief road map of the remainder of the dissertation.

1.1 Challenge

This section explains why the characteristics of the chosen application domain and architecture call for an exceptionally challenging species of load distribution mechanism:

- runtime load distribution rather than compile-time mapping,
- dynamic object migration rather than static object placement,
- load balancing rather than load sharing, and
- global load balancing rather than local load balancing.

The applications motivating this research are object-oriented, with objects dynamically created and destroyed as the program runs in response to unpredictable external events. Therefore, it is not possible to *map* the computation onto the architecture at compile time, as with more regular computations. Instead, a run-time load distribution mechanism is necessary.

Moreover, not only the population of objects but also the pattern of activity is dynamic. Therefore, *static* load distribution schemes which control only the initial placement of computational objects seem inappropriate. Instead, this thesis focuses on *dynamic* load distribution, in which existing objects can be migrated to new processing elements. (A *processing element* or *site* is a unit consisting of one or more processors, memory directly accessible to those processors, and network interfacing and routing hardware. An *ensemble* machine is a collection of processing elements which can communicate via an interconnection network.)

Dynamic load-distribution systems can be divided into two categories according to their fundamental goal: *load sharing* systems attempt only to prevent unnecessary idleness, while *load balancing* systems attempt to ensure that each site has an equal load. Krueger and Livny note in [40] that load sharing suffices to minimize the average waiting time of computations and maximize the throughput of the system, but load balancing may be more appropriate where "fairness" is also an issue. The soft-real-time systems considered in this dissertation are a perfect example of such a situation. What *soft real time* means is that although there aren't hard response deadlines which the system absolutely must meet, the system's performance is judged by its ability to respond to inputs with *consistently* low latency. For example, the system may be updating aircraft positions, headings, and activity descriptions on a situation board viewed by humans, basing the display on received signals. Although there is no absolute maximum delay between the reception of each signal and the corresponding updating of the board, the usefulness of the board will degrade rapidly if the viewers can't count on the entire display to be approximately up to date. This emphasis on consistently low latency rather than merely high throughput makes load balancing more appropriate than load sharing.

Load balancing can offer an additional benefit given our assumptions regarding object-oriented systems. We assume that what is migrated are the objects, that the tasks arise from the reception of messages by the objects, and that these tasks are executed by the processing element on which the object resides. Therefore, migrating an object not only shifts the processing responsibility for existing tasks (i.e. messages already queued for processing by the object) but it also shifts the site at which additional tasks will be created as further messages are received. If an object which has recently received many messages is more likely to receive additional messages, then migrating objects to balance the number of pending tasks may also tend to balance the arrival rates for future tasks.

There are two general strategies for load balancing, known as *global* and *local* load balancing. Global load balancing attempts to estimate the instantaneous average over all processing elements of their loads and then move work from overloaded sites to underloaded ones so as to bring them to the system-wide average. Local load balancing, on the other hand, compares the loads within each of many pairs of sites (or other small groups of sites) without reference to any notion of a system-wide average load, and moves work so as to remove any load imbalance within each pair (or other small group). The work transfers may move work from one overloaded site to another less overloaded site, or from an underloaded site to another more underloaded site, both of which would be avoided by global load balancing. With sufficient repetitions, local load balancing can achieve system-wide load balance, since the pairs or small groups are typically chosen to overlap or are chosen on a dynamic, randomized basis. The disadvantage, however, is that it may require more work migrations to do so. In contrast, global load balancing would in principle migrate any object at most once and would never migrate an object from a site and then another object to that same site; however, it requires global information, which may be expensive or impossible to obtain in a timely fashion. The focus of this thesis is to overcome this difficulty.

Our notion of "global" vs. "local" load balancing is related to but distinct from the notion of network locality. One obvious way to choose the pairs of sites for a local load-balancing system is to use neighboring sites in the interconnection network. When this is done (or other small network neighborhoods are used), the load balancing is "local" in a stronger sense; this corresponds to what Halstead termed *diffusion* load balancing [31]. We will reserve the term "local load balancing" exclusively as a descriptor of the basic balancing objective. Even if work is migrated between distant sites, the load balancing is "local" if the amount of work migrated is chosen so as to equalize the loads of the two sites in question.

For global load balancing, on the other hand, there is little realistic choice but to allow non-neighboring sites as partners for load transfers. Major load imbalances can exist without any pair of neighboring sites being on opposite sides of the system-wide average load.

The relative merits of diffusion, more general local load balancing strategies, and global load balancing are intimately tied to both the hardware architecture and the application domain.

Early ensemble architectures such as Halstead's MuNet [31] provided only nearest-neighbor communications; passing a message to a more distant site required full processor intervention at each intervening site to store and forward the message. Thus the cost of "directly" migrating an object to a distant site would not have been much less than that of diffusing it one site at a time.

In contrast, the target architecture for the work described in this thesis supports cut-through routing, in which messages stream through intervening sites in a pipelined fashion without processor intervention. In this case, the latency of a message transmission is proportional to the sum of message length and transmission distance, rather than their product, in the absence of contention. Thus, strategies involving non-local work transfers are feasible, and in particular global load balancing can be a serious contender.

These architectural considerations are important enabling factors for global load transfer and hence global load balancing, but they are not in and of themselves motivation to use global load balancing. This is where consideration of the application domain becomes relevant. Recall that the primary virtues of global load balancing are that it can prevent repeated migration of the same object to successively less loaded sites and that it can prevent unnecessary migrations of objects from underloaded sites which will receive replacement objects. For an application where throughput is the primary performance measure, these repeated and unnecessary migrations contribute to overhead but do not more directly stand in the way of application performance. On the other hand, in the soft-real-time applications that are the focus of this thesis, the performance objective is consistently short latencies rather than high throughput; excessive migrations can substantially lengthen latencies and thus detract from this objective.

Thus, in order to obtain consistently short latencies in a dynamic, object-oriented, soft-real-time application running on a modern ensemble architecture with cut-through routing, it appears desirable to do dynamic global load balancing. The problem is that although we can rapidly communicate between distant sites, this does not imply that all sites can rapidly obtain information about all other sites, particularly if there are hundreds or thousands of them. Therefore, it will in general be impossible for all sites to determine the instantaneous system-wide average load before it changes. This combination of the desirability of global load balancing with the difficulty of obtaining current global load information is the essential challenge faced by this thesis.

1.2 Opportunity

Having seen that the challenge is to produce accurate estimates of the current system-wide loading without rapid global information sharing, it is time to consider what opportunities exist for escaping this dilemma. If information dissemination takes time, then the system-wide information actually available will reflect historical loads, rather than current ones. Is this of any use? Yes, if there is some relationship between past loads and present ones, such that knowledge of the past loads can contribute to an estimate of the present load. To some extent this will be the case in any computer system—the load doesn't just randomly jump from one level to another completely independent one at each instant—but we will see that it is especially true in the soft-real-time surveillance systems under consideration, because their loads are statistically periodic.

Programs in our chosen application domain are driven by input from periodic sampling or scanning of the real world, with sufficient continuity from period to period to allow tracking of changes. Thus the input arrives periodically with adjacent periods having similar inputs. For example, the number of aircraft observed by a RADAR system will not change precipitously from one scan to the next.

This periodicity of input can translate into a periodicity of loading of the computer system, provided that two conditions are met. First of all, the processing must be largely determined by the input data. For example, if the majority of the load arises from processing triggered when an aircraft newly deviates from its flight plan by more than an allowed amount, then the continuity of inputs may provide little continuity of processing load. The applications under consideration in this thesis do expend some processing effort on such computations, but largely perform more routine processing which is as consistent from period to period as the data.

The second condition necessary for input periodicity to show through as load periodicity is that the system must be sufficiently powerful to keep up with the input, rather than smoothing out the load by building up work backlogs during the peaks for processing during the valleys. For example, many traditional batch processing systems have highly periodic loads but are intentionally sized for the average load rather than the peak load so as to efficiently spread the load out, since the primary goal is high utilization. In real-time computing, on the other hand, the focus is on low latency rather than utilization, so systems are sized for peak loads. Therefore, in our chosen application domain we can expect to see periodic loads.

Note that the periodicity under consideration is of a statistical nature, not a rigid mathematical one. Adjacent periods are similar but not identical, and distant periods may bear no resemblance at all. None the less, the correlation between loads from corresponding points in different periods adds to the correlation between loads at consecutive instants to help make the historical load data more relevant to estimating the current load. This relevance of historical data constitutes an opportunity; the primary thrust of this thesis is the exploitation of that opportunity.

1.3 Contributions

The major contribution of this thesis is a technique whereby explicit stochastic-process models of both loading and information dissemination guide the assimilation of historical load information into improved estimates of instantaneous system-wide load. This is done by applying the statistical techniques of time-series analysis and Bayesian inference. The improved load estimates enable successful global load balancing with its attendant low migration rate. This technique is examined not only analytically but also through empirical simulation studies with an experimental implementation.

On the more theoretical side, this dissertation presents modelling, optimization, and algorithm design contributions. In particular, it analyzes a randomized fully-distributed algorithm for load-metric averaging and uses that analysis to optimize a free parameter of the algorithm. This averaging algorithm has the interesting property of producing repeated estimates of the average with a range of tradeoffs between accuracy and delay. Next this thesis shows how a simple model can be effective at capturing the statistical structure of the time evolution of the load, in particular its periodicity. These two models—one of the averaging, the other of the load itself—are integrated together to allow an optimal weighting to be inferred for the various ages and qualities of load information available. This method of load estimation is combined with other elements to form a complete algorithm for global load balancing. Finally, analysis yields approximations to the improvement this load estimation technique can offer relative to the alternatives of using either only old but complete load data or only recent but incomplete data.

Having presented this load-balancing algorithm (with attendant analysis), this thesis then goes on to explain how it was actually implemented and to present experimental results obtained from simulations using the implementation. The simulation results show not only overall performance impact but also the more detailed effects underlying the system performance. In particular, the load-estimation technique proposed in this thesis is shown to indeed produce improved load estimates. Further, it is confirmed that these improved load estimates allow load balance comparable to that of other methods to be achieved with many fewer object migrations, and that migrations do induce significant application-level latency. When object migration is relatively cheap, the extra overhead of the proposed load-estimation technique counterbalances these factors and results in approximately equal overall performance as with simpler dynamic load balancing techniques. However, the simulation results show that when object migration is more costly, there can be a net performance improvement from using the proposed technique. Finally, additional simulation data provides some insight into how the proposed load-balancing method would fare when used with an application having a highly regular static arrangement of objects, and in particular how various combinations of object placement mappings and object migration techniques interact.

1.4 Outline of the dissertation

- Chapter two describes the architectural and application context of this research. It describes the simulated multicomputer used, the object-oriented concurrent programming language in which the applications were written, and the soft-real-time surveillance applications themselves.

- Chapter three surveys related work. It cites work upon which this dissertation is based and also differentiates the approach presented here from other approaches.

- Chapter four presents our stochastic-process models for the averaging process and for the system loading, shows how those models can be combined, describes our load-estimation and load-balancing algorithms, and presents related analysis.

- Chapter five outlines the experimental implementation of these algorithms. This necessitates addressing various details omitted in the original algorithm description. This chapter also provides an accounting of the simulated time taken by various load estimation and balancing actions, which is important background for the empirical results.

- Chapter six describes the experiments performed and analyzes the data obtained from them. This includes both application performance and more detailed performance-related effects. Although the majority of this chapter focuses on experiments with one of the targeted class of surveillance applications, the chapter concludes with additional experiments performed using a radically different, highly regular, static application, in order to delimit the applicability of our methods.

- Chapter seven concludes by summarizing the results and contributions of this dissertation and by enumerating open questions remaining for future investigation.

Chapter 2

Application and Architectural Context

As stated in the introduction, this thesis focuses on the special combination of challenges and opportunities provided by a particular class of applications running on a particular class of architectures. This chapter describes the application and architectural context of the dissertation more precisely by presenting the specific software and hardware systems which motivated this work and which provided the experimental setting. In particular, the following sections describe the ELINT and AIRTRAC surveillance applications, the LAMINA concurrent object-oriented language, and the CARE simulated ensemble multicomputer. All of these systems were developed in the same research group as this dissertation research was conducted in, the Advanced Architectures Project [53] of the Stanford Knowledge System Laboratory.

2.1 The ELINT surveillance application

Two applications written in LAMINA and running on CARE formed the primary motivation for the work reported in this thesis: ELINT and AIRTRAC. Of these, ELINT was the older and hence more stable program, and thus was used for the majority of experimentation. This section describes ELINT, while the following section outlines AIRTRAC.

The ELINT (ELectronics INTelligence) system was originally designed as one component of the TRICERO multi-sensor information fusion system [62], a sequential uniprocessor program. It was later redesigned to use concurrent replicated pipelines of objects [12], and then yet later a LAMINA version was written, using a similar design [54]. The experiments described in this thesis were based on that LAMINA version, with a few bug fixes and further minor improvements.

ELINT receives pre-processed reports of passively-acquired emissions from RADAR systems and reaches conclusions about the tracks, quantity, nature and activities of the aircraft whose emissions are observed. The system's activity is data-driven, with the input supplied periodically by two or more potentially mobile observation stations. Because ELINT is intended to drive a situation-board for human assessment, there are no hard constraints on its latencies, but they should only infrequently be longer than normal, and then not by much, lest the situation board becomes unreliable.

The input to ELINT consists of a time-ordered stream of observation records, where each observation contains the following information:

- the time at which the observation was made, quantized to an integer input period,
- the observing station making the observation and its current location,
- an integer track number for the observation; the pre-processing equipment attempts to assign a single track number to all observations of a single emission based on signal characteristics and continuity, but may not always succeed,
- the direction in which the emitter lies, relative to the location of the observing station,
- the type of emitter (e.g. missile guidance, navigation, or identify friend or foe), based on the characteristics of the signal, including when known the specific equipment,
- the mode of operation of the emitter (e.g. searching or locked-on) for those with multiple modes, and

- a general indication of signal quality.

Based on these observations, ELINT computes the fix (position) and heading of emitters, associates into a cluster any emitters which remain together, makes assessments as to the nature of the aircraft in each cluster based on the emission types, and identifies immediate threats (e.g. RADARs locked on targets). Reports are generated each input time period containing the results of these computations; the intention is that these reports would be used to update a situation board display.

The LAMINA implementation of this program uses objects to model the emitters and clusters of emitters. Hence objects are created and destroyed at runtime in response to the data, and thus in this context even "static" load balancing means creation-time placement rather than compile-time partitioning. Another noteworthy aspect of the LAMINA implementation is that each real-world object is typically represented by several LAMINA objects. For example, each emitter is represented by a four-deep pipeline of objects, each of a specialized class dedicated to one aspect of the processing (observation processing, fix determination, heading computation, and clustering status control). Thus there is a greater opportunity for excessive object migrations to lengthen individual output latencies than would be the case in the absence of this pipelining. Figure 2.1 shows the overall organization of the LAMINA implementation of ELINT.

To give some indication of granularity, most method invocations take 200–600μs, though a few are as brief as 40μs or as long as 1.7ms. The object states and messages are typically several tens of words in size.

2.2 The AIRTRAC surveillance application

Another soft-real-time surveillance application, AIRTRAC [45], was under development concurrently with this research, and provided additional evidence of the applicability of the load modelling process described in this thesis. However, because it was in a state of flux and earlier versions were not kept current with changes in LAMINA and CARE, it was not possible to use it in the actual load balancing experiments. Therefore, it only merits brief comment here.

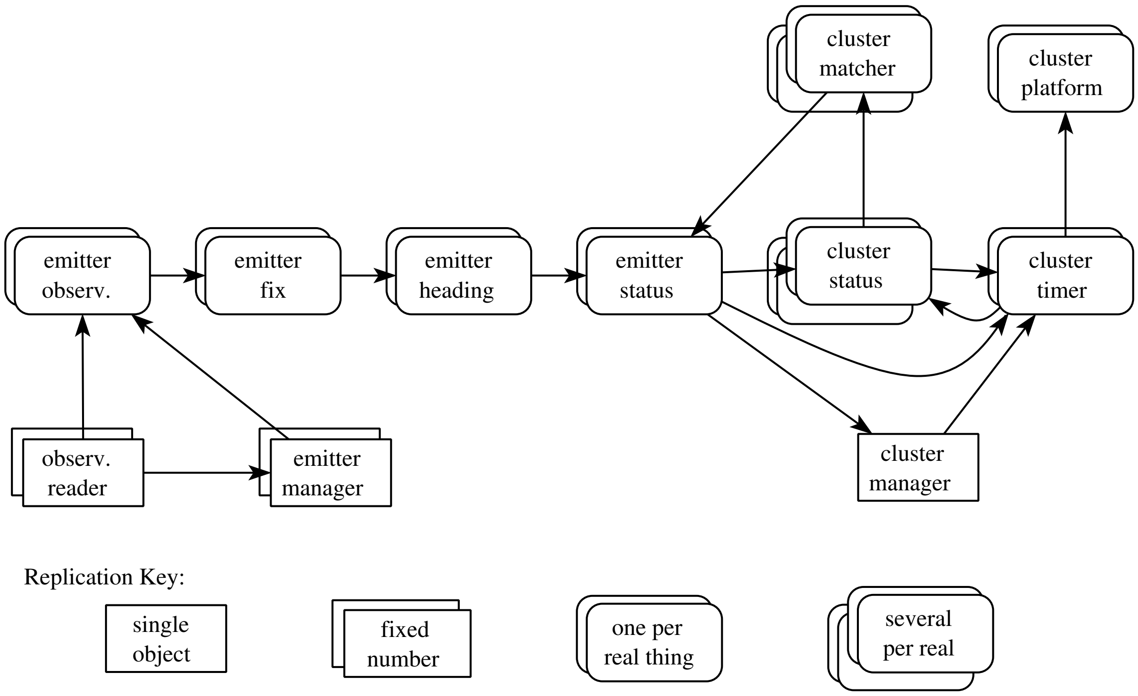

Figure 2.1: ELINT organization. The labels in this diagram are the names of general classes, but the arrows actually show communications patterns between specific instances of those classes. The different box shapes indicate the number of instances of each class. For example, there is one cluster manager, a fixed number of emitter managers, one emitter fix per real-world emitter, and several cluster matchers per real-world cluster. This figure is adapted from [54], which also describes the function of each class and the nature of the messages they exchange.

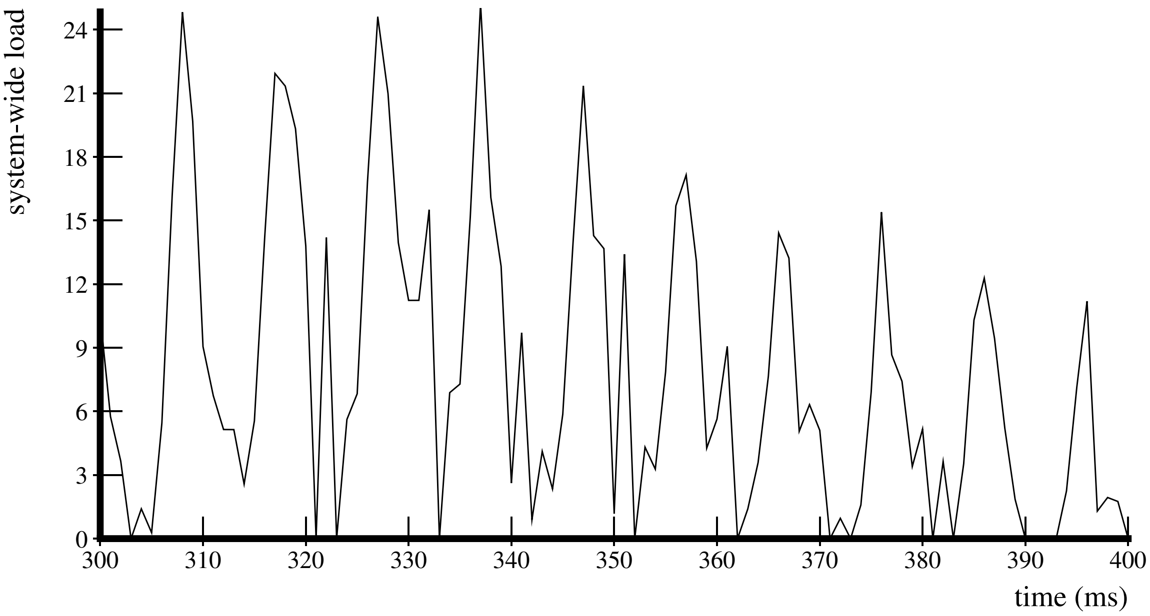

Figure 2.2: A load example from AIRTRAC. Note that although each period resembles neighboring periods, the load pattern gradually shifts.

The AIRTRAC problem consists of using active RADAR tracking and acoustic sensors to characterize aircraft flight paths, with the goal of identifying flights to or from airfields used by smugglers. As in ELINT, the majority of the work occurs in a data-driven pipelined fashion. The details are rather different, however. For example, the active RADAR already provides position and heading (as well as covariances of them), and there is no need to do clustering. However, instead of spatially clustering emitters, it is necessary to temporally cluster observations into flight path segments of approximately constant velocity as the first step in characterizing the flight path.

Figure 2.2, which is reproduced from [30], shows a typical section of a load trace observed in the earlier load-modeling experiments with AIRTRAC reported in that paper. This figure illustrates the form of statistical load periodicity exploited by this thesis.

One interesting experience with AIRTRAC and load balancing is reported by Nakano and Minami in [45]. They initially used a somewhat sophisticated static load-balancing scheme that attempted to spread objects evenly within limited neighborhoods of the creating object. The result was dismal performance, which was improved greatly by switching to random placement. This helps motivate the choice of random placement as a baseline in the experiments reported in this thesis. However, Nakano and Minami go on to report that uniform random placement was further improved upon by reserving sites for key static objects with heavy loads; this is indicative of the sort of thing dynamic load balancing might do automatically, by migrating other objects off of the sites holding those heavily used objects.

2.3 The LAMINA object-oriented system

The surveillance applications motivating this thesis were programmed for CARE using the LAMINA language [21, 19]. LAMINA is a concurrent object-oriented extension to the LISP programming language. A LAMINA application consists of a collection of independent, concurrently executing objects which are instantiations of general object classes. The objects communicate by asynchronously transmitting messages, which trigger execution at the receiving object. In this regard, LAMINA is descended from Hewitt's Actor model [1].

LAMINA is not a purely object-oriented language; in addition to objects and scalars such as numbers, there are structures such as pairs and arrays. (There are also so-called *streams*, which will be described below.) The methods associated with an object can mutate both that object's instance variables and also the structures referred to via those instance variables. In particular, since structures are mutable (and can recursively contain structures), they can form arbitrary directed graphs. However, these mutable structures are local to individual objects; when a structure is passed in a message, the graph rooted there is copied to form an isomorphic, independently mutable graph.

One unusual feature of LAMINA is that the "mailboxes" at which messages are received are first-class entities independent of objects. These mailbox-like entities are termed *streams*; they are the only shared mutable entity in the LAMINA system, and the only one passed by reference rather than by copying. Each object has a primary task stream, and typically loops repeatedly reading a message from that stream and invoking the method specified in the message. However, there is nothing to prevent an object from reading from a different stream, which it may have created for the purpose or may have received a reference to in a message. Similarly, although typically only one object will read from a stream, this is not mandatory. Streams (including objects' task streams) may be configured to present queued messages in priority order rather than FIFO, using priority tags included in the messages. The same priority tags can also be used as sequence numbers, in which case the object is only invoked with the smallest-numbered message if its predecessor has already been processed. Finally, streams can be set to forward their messages to other streams; this capability is exploited by the object migration mechanism.

The only form of atomicity provided is that each method invocation is locally atomic, in the sense that no other method on the same object will be concurrently invoked. However, methods on other objects may execute concurrently. Further, message sends are asynchronous, with the reply (if any) constituting a separate independently atomic invocation.

The implementation of LAMINA on CARE always stores the entire state for each object on a single site, executes all methods of that object on the same site, and queues on that site unread messages of streams most recently read by that object.

2.4 The CARE multicomputer

The CARE multicomputer architecture [19] is a message-passing machine containing on the order of 100–1000 processor-memory sites, each with an additional special purpose processor for housekeeping chores and a dedicated router to allow direct cut-through communications.

The main processor at each site, called the *evaluator*, is a general purpose processor which executes the application code. It is the evaluators' loading which our load-balancing system attempts to improve.

The simulation does not contain a detailed model of the evaluators; instead, the application code is run directly on the simulation host and the resulting time scaled appropriately. Care was taken to make the timings as repeatable as possible by accounting for paging and other low-level activities, in order that controlled experimentation would be possible. This proved inadequate, however, so for the experiments reported in this thesis a different technique was used. The scaled timings were logged in one preliminary run with each time labeled with the invoked method name and object class. These times were then averaged for each method/class pair. The experimental runs were rigged to pretend that each method invocation had taken the corresponding average time. This allowed controlled observation of the effect of load balancing, but may have made the simulation less realistic. However, no qualitatively different behavior was observed than when the actual scaled timings were used, and the times logged in the preliminary run did not show any major variation other than by class and method. The load balancing algorithm was designed on the realistic assumption that execution times were not known in advance, even though in our experiment the time taken by a pending task could in fact have been deduced from the message tag and the receiver's class.

The special purpose processor, called the *operator*, implements the load-balancing algorithm as well as providing message encoding and decoding, message queuing, and process scheduling. (Message encoding is the process of converting pointer-based structures to a compact linear form; decoding is the reverse. Sections 2.3, 5.5, and 5.6 amplify on the need for encoding and decoding.) The operator has a tight shared-memory coupling to the evaluator. One of its intended responsibilities, garbage collection, was not implemented in the simulation. The simulation model of the operator accounts for the time taken by each operation by totalling up times for component actions; chapter 5 specifies the operator times for load-balancing actions.

The communications network is a high-performance cut-through network with a bidirectional toroidal mesh interconnect and hardware support for multicast [13]. It pipelines multi-hop transmissions, even in the multicast case. The network is simulated at the register-transfer level; the simulated times therefore correctly reflect contention effects. Several routing algorithms are available; in the experiments reported in this thesis, the routing algorithm chooses the route dynamically at each site so as to avoid unavailable links, provided the message moves strictly towards its target. If all communication links in the direction of the target site are unavailable, the message is delivered to the local operator's queue and then later re-transmitted by that operator.

From the above it is apparent that CARE is structurally similar to contemporary ensemble machines, such as the Intel Paragon [65]. As in CARE, each node of the Paragon contains a router, memory, and two processors—one for communications, the other for computation. Further, the Paragon's nodes are connected in a two-dimensional mesh, just as in CARE. However, one difference between CARE and contemporary ensemble machines merits note: CARE is several times slower, reflecting the state of hardware technology at the time this research was begun. Although it is possible to smooth over these shifts of technology by citing times in clock cycles, rather than nanoseconds, in this thesis we will use actual times. There is little reason to suppose that the speed not only of all the hardware sub-systems, but also of the external data source, would scale at the same rate. The concluding chapter provides some suggestions as to how the results of this thesis might be interpreted within the context of more modern systems.

2.5 Context summary

The applications for which our load-balancing method is intended are driven by periodic observations of an external situation that dynamically varies. The programs dynamically create software objects to correspond with observed entities in the external world, such as emission sources; typically several software objects will be created for each real-world entity. The objects vary in role within the data processing; for example, there are key "manager" objects. Outputs are produced as the result of pipeline-like flow of data through multiple objects; the latencies of these outputs should be kept consistently small. The state of each object resides in the local memory of a single processing element of the ensemble machine, and all processing for that object occurs at the same site. The sites are connected with a cut-through mesh network that supports multicasts. An extra processor is available at each site for such tasks as message processing and load balancing.

Chapter 3

Related Work

This chapter surveys prior work with two distinct goals: differentiating our load-distribution problem from alternative problems, and exploring the similarities between our work and that of others.

First, we will elaborate on section 1.1 by more carefully positioning our particular load-distribution problem within the full taxonomy of such problems. Since the goal here is to position our thesis in this taxonomic tree, we will concentrate on stating the essential distinctions between the various branches of that tree, rather than on describing the work that has been done in the other branches.

Having differentiated our problem from other problems, we will examine prior work more closely that is similar to ours in one way or another. What have others done who were faced with real-time applications? What other work has a similar emphasis on employing uncertain, out of date load information? What mechanisms similar to ours have been employed for the underlying tasks of disseminating load information and migrating communicating objects?

3.1 A taxonomy of load distribution problems

The general problem of how to divide a workload among multiple processors comes in many variations, illustrated in figure 3.1 and described in this section. The highest level distinction is between approaches such as ours that distribute the data (whether

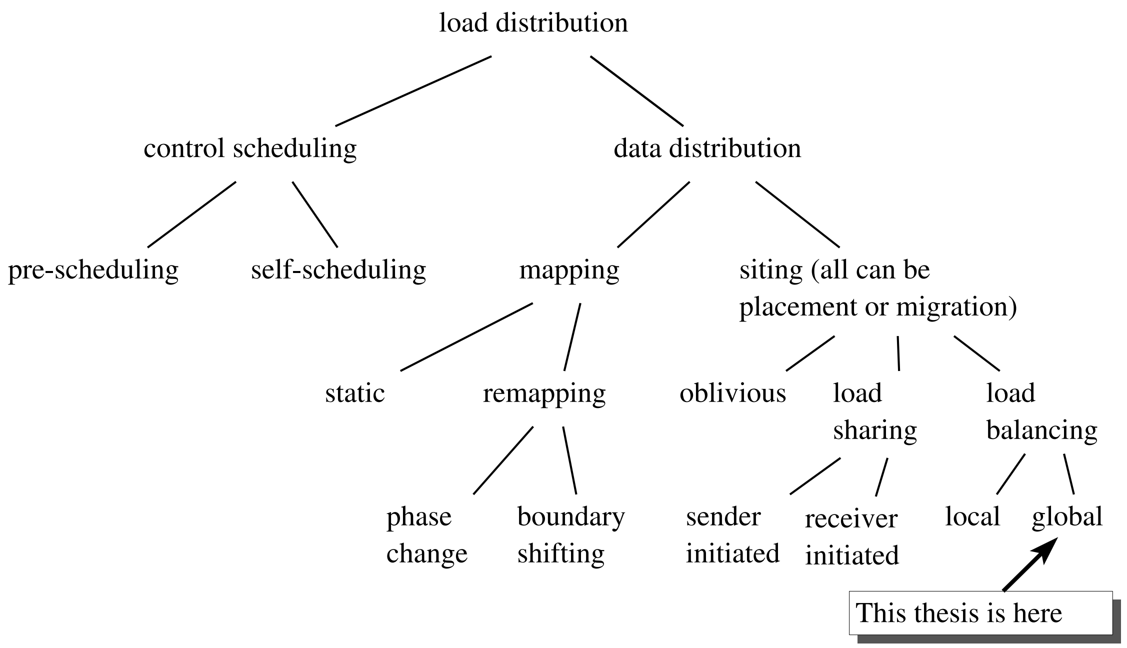

Figure 3.1: Load distribution taxonomy. This diagram shows the varieties of load distribution problems, illustrating the degree to which our problem of global load balancing is related to each of them.

at the level of entire process states, objects, or individual array elements), and those which attempt to schedule elements of the control flow (such as loop iterations), leaving the questions of data location and access implicit. Control scheduling is sufficiently remote from the approach taken in this thesis that we will only expand upon the data distribution side of the taxonomy in the following subsections.

Within this data distribution side of the taxonomy there is another dichotomy. Some researchers have focused on the structure of the data, e.g. as a regular two-dimensional array or a particular graph structure of interconnection among communicating processes. The problem is then conceived of as how to systematically map this structure onto the physical structure of the hardware. Others treat each unit of data independently, with the problem being to separately choose an appropriate processing element for each. The work presented in this thesis falls into the latter category, because the structure of communicating objects in our target applications is dynamically generated and constantly changing. However, it is worth briefly mentioning some highlights of the alternative approach (mapping), for the sake of context.

3.1.1 Mapping

Within the general mapping approach, a primary distinction is between those variants that do the mapping once and for all in advance, and those that remap at run-time to accommodate changes in the amount of computation being done on various parts of the static data structure, or to accommodate other changes, such as in the relative proportions of computation and communication.

Static mapping

The primary virtue of static mapping is that resource scheduling is not competing for the same computational resources as the application program. However, there may be some sacrifice of performance, because the load distribution can't be adapted to changing conditions. In particular, if the locus of computational activity shifts unpredictably, the mapping has to be designed to perform well regardless.

One typical example is a message-passing implementation of the Zipscreen battlefield simulator done by Nicol [46, 48]. The simulation takes place on a two-dimensional hexagonal grid; units of two opposing forces move from hexagon to hexagon and engage in combat when they arrive either on the same hexagon as a unit of the opposite force or on a neighboring hexagon.

Because computation is concentrated in the area of active battles, which tend to be geographically localized, there is a strong correlation between the loads associated with neighboring hexagons. Therefore, good load balance can be achieved using a wrapped (or modular) mapping, in which neighboring hexagons are assigned to neighboring processing elements; that way each hexagon in any region smaller than the size of the ensemble is guaranteed to be on a distinct processing element. However, since the simulation involves interactions of units in neighboring hexagons and movement of units from hexagon to hexagon, this wrapped mapping produces considerable network traffic. A block mapping, in which a single contiguous block of hexagons is assigned to each processing element, with neighboring blocks on neighboring sites, minimizes this communications traffic, but at the cost of poor load balance (since an entire battle may take place on a single processing element).

Nicol's solution is to strike a balance between the two; the domain is partitioned into small contiguous blocks (many more than there are processing elements), and then the blocks are assigned to the processing elements in a wrapped fashion. That way, if the blocks are smaller than the areas of intense activity (battles in this particular application), some load balance is still achieved by spreading the activity across multiple sites, but a network message isn't necessary for each crossing of a hexagon boundary. Nicol and Saltz [48] also applied this same approach to other problems with shifting, localized workloads, such as a fluid dynamics code with adaptive gridding. Note that although these problems have irregular workloads, there is an underlying regular structure; this distinguishes them from our target applications, where there is no fixed structure to aggregate or map.

Remapping

Nicol and Reynolds [47] use the same two examples, Zipscreen and the adaptive gridding fluid dynamics code, to illustrate one possible application of remapping. In a typical Zipscreen simulation, there may be a phase where opposing forces move towards one another, followed by another phase where they actively do battle. The first phase is communications intensive, and would benefit from a mapping with large blocks, while the second phase is computationally intensive and would benefit from a smaller block size. Similarly in the fluid dynamics code, there may be little adaptive gridding until a shock wave forms, and hence it may not make sense to use a fine-grained mapping until that point. Nicol and Reynolds provide a heuristic for deciding at what point to switch between the two mappings, based on unreliable evidence as to whether the phase change has occurred. Their experimental results show that the resulting performance is nearly as good as an optimal choice between the two mappings at each time step.

The work of Cap and Strumpen [14] exemplifies an alternative remapping strategy, designed to cope not with phase changes but with evolving imbalances in the computational load on the various sites. This technique is to do an initial block mapping (or the one-dimensional analogue, a strip mapping) and then periodically shift the block boundaries so that heavily loaded sites loose some of the data domain to more lightly loaded neighbors. This is rather reminiscent of the diffusion load balancing technique (defined in section 1.1), but continues to presume a definite systematic mapping of a regularly structured data domain onto the structure of the ensemble machine. Diffusion, by contrast, fits into a different branch of our taxonomy (shared by our own work), because it presupposes no structure on the collection of independent objects and can wind up siting them in any arbitrary arrangement on the processing elements.

3.1.2 Siting

At this point, we've examined alternative load distribution problems that differ from ours at the most fundamental level, by distributing control structures rather than data structures, and at the second level, by systematically mapping a structured data domain onto the hardware structure rather than independently siting individual data units. This brings us again to the category containing our own work, namely load-distribution problems concerning the siting of individual entities (typically either objects or processes).

Alternative forms of the siting problem can be differentiated in two orthogonal ways. One, which is shown explicitly in our taxonomy diagram (figure 3.1, page 20), consists of a closely related pair of issues: the load distribution objective and the information used to achieve that objective. It is in this dimension that our own problem of global load balancing is differentiated from local load balancing, load sharing, and completely oblivious siting. The other, orthogonal, distinction is that siting can be done exclusively by placing newly created entities or can encompass migration of existing entities as well. Thus our work, which focuses on migration, is also distinct from all techniques confined to placement (though typically those are also not global load balancing).

As explained in section 1.1, one fundamental difference in siting approaches is between those such as ours that attempt to actually balance the load, and those that merely attempt to prevent unnecessarily idle processors (load sharing). However, neither of these two categories adequately encompasses the very simplest siting policies, in which entities are sited in a manner completely oblivious to loading, e.g. randomly. Although such systems could perhaps be better labeled as load balancing than load sharing, it seems better yet to put them in a class of their own, since oblivious policies aren't closely tied to either objective.

Oblivious siting

Recent interest in oblivious siting schemes has focused on random placement; however, it is also possible to do the placement deterministically, or even to do oblivious migrations. Wang and Morris [61] in their survey of early load siting literature include two references to work on cyclic placement, i.e., deterministic, oblivious placement. Although this approach is useful when a single source of work is doing the placement, it is ill-suited to systems such as ours in which the placement needs to be done on a decentralized basis. We are unaware of any studies of random migration, presumably because oblivious siting provides no reason to expect any randomly chosen migration to be beneficial. However, in principle it would be possible to do random migrations at some chosen rate; this would be somewhat analogous to stirring a pot to ensure that it stays homogeneous.

Athas [5] advocated random placement of fine-grained objects on ensemble machines, but provided only limited empirical evidence showing that it provided good speedups for some simple benchmark programs when communication costs were ignored, and that it produced predictably poor locality of communications. Grunwald's thesis [29], in contrast, provides a more thorough simulation study comparing random placement with a wide variety of local and global load balancing placement schemes. His simulation accounted for communication costs in a circuit-switched interconnection network as well as processor time. Grunwald's performance metric was the total execution time for each of a range of benchmark programs, synthetic and otherwise. He found that random placement was often the best performer or nearly so, never a particularly poor performer, and frequently much better than many of the other placement strategies.

Comparing those cases where random placement worked well and those where it didn't work as well, Grunwald concluded that it was "an acceptable scheduling strategy if the average process computation time is short, relative to communication delay. For processes using more resources, status and load information is generally beneficial." [29, p. 181] Grunwald's work was concerned with process placement, rather than object placement, but presumably similar results would carry over: for frequent placement of short-lived objects, random placement may be adequate, while for the less frequent placement of long-lived objects, more careful choices are necessary.

These considerations feed into our own work in two ways. On the one hand, Grunwald's relatively successful experience with random placement explains our choice of it as a baseline for comparison in our experiments and as the placement strategy with which our own migration strategy will be coupled. On the other hand, his comments regarding long-lived processes (or, we presume, objects) help motivate our decision to add a more information-intensive migration policy.

The objects in our target application are long lived, since some of them correspond to the comparatively slow-moving entities that inhabit the non-computational world and the others are permanent "manager" objects. Thus it isn't surprising, in light of Grunwald's results, that our experimental results in chapter 6 show that random placement leaves ample room for improvement. We could have followed Grunwald's lead and concluded that in the presence of such long-lived entities, we needed to make more careful, informed placement choices. Instead, we've chosen the more radical path of implementing object migration, since our objects are both very long lived and subject to dynamic changes in their activity level.

Load sharing

Turning to load siting policies that make siting decisions based on a particular performance goal, we encounter not only load balancing systems such as ours, but also load sharing systems that try to transfer work only to prevent unnecessary processor idleness. The two assumptions underlying load sharing are that the units of work being allocated are independent tasks, and that the performance objective is to minimize the mean response time of the tasks. Papers on load sharing haven't always made this clear; an influential paper by Eager, Lazowska, and Zahorjan [25] declares in the introduction that "we consciously use the phrase 'load *sharing*' rather than the more common 'load *balancing*' to highlight the fact that load *balancing*, with its implications of attempting to equalize queue lengths system-wide, is not an appropriate objective." [25, p. 54] Not an appropriate objective? Why not? Only four pages later, in the final section of the paper (where they present performance results) do the authors bother to remark that "average task response time as a function of load will be our measure of performance." [25, p. 58]

Average response time will be minimized by *any* execution schedule that avoids unnecessary idleness; individual tasks can be treated arbitrarily unfairly without this performance metric suffering, because these authors also assume that the tasks are independent. This is completely unlike the situation in our target applications. Firstly, our tasks are coupled by the pipeline-like nature of the processing; delays early in a pipeline will delay all the later components of the pipeline. (The fact that our "pipelines" actually sometimes feed back into themselves, forming cycles, only makes matters worse.) Secondly, we are in any case not satisfied with merely minimizing mean response time, since the nature of the application demands consistency of response times. Similar points were made by Krueger and Livny [40], who compared load balancing with load sharing. They concluded that the choice of policy depends on the users' performance goals as well as the workload characteristics. Load balancing, they reported, has the ability to improve a wider range of performance metrics, including in particular those that attempt to measure fairness.

The primary distinction among load sharing approaches, shown in our taxonomic tree, is between receiver-initiated and sender-initiated schemes. In a receiver-initiated scheme, a site that is idle or in danger of soon becoming idle seeks work from other sites. In a sender-initiated scheme, heavily loaded sites seek lightly loaded sites to which to transfer work. Eager, Lazowska, and Zahorjan [25] point out that the relative efficiency of these two approaches is tied to the overall system loading; in a lightly loaded system it is easier to find a lightly loaded site than a heavily loaded one, while in a heavily loaded system the reverse is true. For those who want to pursue the load sharing approach further, some other interesting papers are [24, 38, 17, 63, 57, 41]. In load sharing, as in many other areas of distributed systems research, attention has lately turned to the additional complexities introduced by *heterogeneity*; see for example [64].

All the load sharing work described above was in the context of distributing user jobs on a local area network of workstations. However, there is one more unusual member of the load sharing family that was proposed for use in an early fine-grained ensemble-machine implementation of a concurrent functional language. Lin and Keller [43] designed their gradient model for use with a store-and-forward grid interconnection network, so they restricted themselves to local transfers of information and tasks between adjoining sites, as in diffusion load balancing. Within this limitation, they implemented a receiver-initiated load sharing system.

Each site uses information from its neighbors to maintain an estimate of its distance from the nearest nearly idle site; this estimate is called the site's "proximity." When a site's load drops dangerously low, it sends out a request for work to its neighbors in the form of a message that its proximity is now zero. Sites which are themselves nearly idle ignore this information, since it doesn't affect their own proximity of zero. Sites with abundant tasks respond to the work requests by periodically transferring tasks to their lowest proximity neighbor. The remaining sites effectively pass on the work request to their neighbors; they do this by recomputing their own proximity (as one more than the smallest of the neighbors' proximities) and notifying their neighbors if this is a change.

Kalé and Shu [37, 55] used simulation to compare the gradient method with their own local local balancing method (described below) using benchmark applications and architectural assumptions which were both quite similar to Lin and Keller's own. They found that the gradient method was insufficiently "agile" at spreading the load, resulting in poor performance. From our vantage point, this result is predictable, since they were comparing a load sharing algorithm with a load balancing algorithm. What is remarkable is that Lin and Keller advanced a load sharing algorithm at all for such an interdependent collection of tasks as the various sub-goals of evaluating a functional program.

Load balancing

The only remaining classes of load distribution approaches identified in our taxonomy are local load balancing and global load balancing. As described in section 1.1, local load balancing is attractive because it does not require global load information and because it is well suited to situations in which load transfers are restricted to being between neighbors. This restriction to neighbors (what we broadly term the diffusion style of load balancing) is particularly attractive in store-and-forward networks; however, even in cut-through networks, there will be some performance benefit from retaining locality of communication.

Local load balancing Halstead and Ward [31] describe an early implementation of a diffusion migration method, in the context of their MuNet store-and-forward ensemble machine. As implemented, an arbitrarily chosen object is migrated between two neighboring sites whenever one is more loaded than the other. However, Halstead and

Ward suggest as an unimplemented improvement that an attractive force for locality be added to the repulsive force for load balance. They speculate that this will improve performance, but provide neither empirical evidence nor any mention of the possible effects on system dynamics. They suggest the metaphor of mutually repulsive bodies connected by springs; to our mind, this immediately brings up images of wavefront propagation, perpetual oscillation, and chaos. Halstead and Ward seem immune to these fears. They do, however, admit that it is an open question "whether reasonable scheduling strategies can be built around the philosophy of decision-making on the basis of local information only, or whether more global information and interaction is required." [31, p. 144]

Hudak and Goldberg [34] experimented with such a more sophisticated attraction/repulsion diffusion strategy in the context of combinator reduction. Their algorithm didn't do any migrations, but instead only determined whether to place a newly created combinator graph node on the creating site or on one of its neighbors. As such, the only actual long distance "diffusion" that occurs is in the sense that a causal chain of node creations may progressively move across the torus network one site at a time.

Hudak and Goldberg's scheme places the new node on the candidate site with the smallest value of a cost metric, calculated as the sum of the site's load plus a weighting factor times the sum of the distances from that site to the sites containing the nodes adjacent to the new node in the combinator graph.

Hudak and Goldberg don't present any results concerning the system dynamics obtained with various values for the weighting factor, but rather only the resulting overall speedup achieved with weighting factors of 0, .2, .4, .6, .8, and 1.0. They focussed on the fact that the highest speedup achieved (when the weighting factor was .6) was only a few percent higher than with a weighting factor of 0 (i.e., pure diffusion). In doing so, they overlooked the far more interesting fact that when the weighting factor was 1, the speedup was 25% lower than the nearly constant level it had for weighting factors of 0 through .8. What is it that happens when the weight given to a task worth of load is exactly equal to the weight given to a hop worth of distance? There is too little evidence in Hudak and Goldberg's paper to tell, but it seems likely that some qualitatively different dynamic behavior occurs in this case.

More recently, Uehara and Tokoro [59] proposed a similar attraction/repulsion diffusion scheme for object migration; in their system, the attraction is computed by averaging a collection of vectors, one for each of the N most recently received messages for some constant N. These vectors point in the direction of the messages' senders and have as their magnitude the time the messages spent in transit. Although Uehara and Tokoro pay nearly as little attention to dynamics as Hudak and Goldberg did, they do remark briefly that they have observed in their simulation that "the system may become instable because of thrashing." [59, p. 112]

Kalé [37, 55] attempted to overcome two problems he perceived in the diffusion approach. One is that there is no limit to the distance a task can move; in fact, in the presence of a shifting load distribution, a task could be passed around for ever. The other problem is that a prospective task exporter may be dissuaded by the comparatively high loads of its immediate neighbors, even if there are any number of desperately underloaded sites just a little further off. To address these problems, Kalé's contracting within a neighborhood (CWN) method stipulates a minimum and maximum number of times each task can be transferred. Until the minimum number of transfers has been reached, the task is passed to the least loaded neighbor, even if that neighbor is more loaded than the current site. Once the minimum number of transfers has been done, additional transfers up to the maximum can continue provided the task now moves only to progressively less loaded sites. Kalé proposed using this technique for the initial placement of tasks, rather than subsequent migration.

Returning to the question of dynamics, it is worth noting that the information on the load of neighboring sites may be out of date; Kalé [37, p. 9] tells us that "this information is maintained by broadcasting a very short message to all neighbors periodically, or as an optimization, piggy-backing the load information 'word' with regular messages." Shu and Kalé [55] show that with a long interval between transmissions of load information the system can behave unstably.

Another general approach to local load balancing, suggested by Ferguson, Yemini, and Nikolaou [27, 26], is to treat it as an economic problem, in which resource allocation is done by independent actors each attempting to maximize their own utility.

Processors hold auctions, in which processes compete for communications and computation resources. This sets a "going rate" for each resource, which will be higher in heavily loaded regions. A process can decide whether it is cheaper to stay on an expensive (i.e., heavily loaded) site or to pay the communications cost to migrate and in return obtain cheaper processing elsewhere. Waldspurger, Hogg, Huberman, Kephart, and Stornetta [60] describe experiments with a task placement system called Spawn based on this premise. Spawn is designed to place coarse-grained processes in a distributed network of workstations. Its designers only tested it using independent tasks on very small networks; under those limited circumstances it performed well.

Global load balancing We've already identified in section 1.1 some of the reasons to consider global load balancing as an alternative to local load balancing. Additionally, Grunwald [29, pp. 145–148] shows that if an application has a workload that rapidly creates a large number of processes, local schemes will be unable to keep pace. In particular, his workload of this form achieved speedups in the neighborhood of 70 on 256 sites with global load balancing schemes (even global random placement), but only around 40 with even the best local load balancing scheme (a variant of CWN).

The simplest way in which global load balancing can be applied is in small, tightly coupled machines, such as small shared-memory multiprocessors. Majumdar and Green [44] did this in a real-time Ballistic Missile Defense (BMD) application. However, there is so little problem obtaining up-to-date system-wide information in this context that the issues considered in this thesis are moot.

Another class of architectures for which global load balancing has been proposed, precisely because global information dissemination is easy, is broadcast-bus based local area networks (LANs) of workstation computers. Although LAN topologies have lately been becoming increasingly complex, at one time it was typical to have many computers connected to a single Ethernet coaxial cable; in that context, global information dissemination and global balancing seemed promising.

One elegant solution proposed by Krueger and Finkel [39] integrates the estimation of the system-wide average load with the process whereby an overloaded site locates an underloaded site to which load can be transferred. This property is shared by our global load balancing method as well; however, Krueger and Finkel's scheme makes essential use of the broadcast nature of the network, while our scheme is tuned to networks supporting efficient multicast but not broadcast.

In Krueger and Finkel's scheme, a site with above-average load (relative to the current estimate) broadcasts this fact. If an underloaded site responds in a reasonable time, a work transfer is arranged. Otherwise, the estimated average load must have been too low, since there was a site above it but none below it. Therefore the site that had believed itself to be overloaded broadcasts a new, higher estimate of the average load. Conversely, if a site believes itself to be underloaded but hears no broadcasts from overloaded sites, it will broadcast a new, lower estimate of the average.

As in our implementation, Krueger and Finkel include an acceptable range around the estimated average to prevent thrashing; only sites that differ from the estimated average by more than a threshold amount take part in the process described above. Krueger and Finkel show by simulation that their scheme improves fairness as well as overall performance, as would be expected from a global load balancing system.

An alternative global load balancing system for similar LAN settings has been proposed by Baumgartner, Wah, and Juang [9, 35]. Their system takes even more detailed advantage of the nature of LANs such as coaxial-cable Ethernet; in particular, these networks not only are physically a broadcast medium, but moreover they resolve contention for this shared communication channel by collision detection. That is, it is possible (in fact necessary) to determine whether other sites are attempting to transmit simultaneously.

Baumgartner, Wah, and Juang note that only a single task can be transferred at a time on each such "contention bus" (in [35] Juang and Wah consider the case that several parallel busses each connect all the sites). Therefore, there is the special challenge for global load balancing to not merely transfer between overloaded and underloaded sites, but in fact at each moment transfer from the most loaded site to the least loaded site (or from the most loaded n to the least loaded n where there are n busses).

In the spirit of our own thesis, Baumgartner, Wah, and Juang tackle this special challenge by exploiting a special opportunity presented by contention busses; they provide protocols that use collision detection to efficiently locate the most and least loaded sites. Inspiring though this approach is, it does not solve the problem to which this thesis is addressed. We face our own, different combination of challenges and opportunities, and will need quite different techniques as a result.

Turning to networks such as CARE's, the only prior mentions of global load balancing we are aware of other than Grunwald's are arguments as to why it is impractical in this context. Athas writes in his thesis [5, p. 74]:

> Unfortunately, for ensemble machines at least, data about the state of the computation is localized to each node and possibly the set of neighboring nodes. Each node must make decisions about placing objects based on limited and perhaps stale information about the state of the other nodes in the ensemble. .... Obtaining accurate global information about the state of the computation is not practical.

Grunwald implemented global load balancing using simulated broadcast in binary hypercube ensembles of size up to 256 sites. His experimental results generally seem to confirm Athas's pessimistic assessment, in that only rarely was the performance significantly better than with random placement (and often much worse). However, it is important to note that he used brute-force information dissemination, rather than the more sophisticated load estimation techniques proposed in this thesis, and used only initial placement, rather than migration as in this thesis.

This is the gap in the taxonomic tree where we endeavor with this thesis to send forth a new shoot: we will attempt, for the first time, to develop sophisticated techniques for estimating global loads (rather than merely communicating them) so as to permit successful global load balancing in a large-scale ensemble machine connected by a low-degree network (i.e., one in which each site is directly connected to only a few others).

3.2 Similarities to prior work

3.2.1 Real-time systems load distribution

Most prior work which shares with us the real-time application domain has been quite dissimilar in other regards. For example, we cited above a real-time BMD application [44] that did global load balancing, but was quite different than our work because it presumed a tightly-coupled shared-memory multiprocessor. Similarly, another BMD-related project [32], the Radar System Test Driver of Huang, Shih, and Machleit, resolved the load distribution problems primarily through static database partitioning, which is inappropriate for our target applications.

Kurose and Chipalkatti [42] performed analytical modeling analogous to the above-cited load sharing study by Eager, Lazowska, and Zahorjan [24], except that a real-time performance metric (percentage of jobs missing their deadline) was used. As with the earlier study of load sharing, an assumption was made that the tasks being distributed are completely independent. Thus, even though good miss rates were shown to be achievable with simple load sharing algorithms, there is little relevance to our domain of tightly interdependent communicating objects. Ramamritham, Stankovic, and Zhao [50] make a similar assumption of task independence; thus, their simulation work suffers from the same limited relevance to our own application domain.

One particularly interesting real-time load distribution study is Chang's thesis [15], which is in the comparatively similar context of process migration on a LAN. Most of that study focusses on independent tasks, but a portion assumes that the tasks form pipeline structures. However, that portion of the thesis is limited to placement rather than migration, to priority maintenance rather than deadline observance, and to the case of only two priority levels.

Chang's thesis focusses on the problem of choosing which tasks to migrate; thus, it is best seen as complementary to our own work, which largely neglects this topic. Chang's contribution to the task selection problem consists of two heuristics, which he terms triage and global priority. The former is intended to reduce the fraction of tasks that miss their deadlines, while the latter is intended to assure preferential assignment of processing resources to high-priority tasks.

In the triage approach, tasks are moved if they both can't meet their deadline if left on their current site and can meet their deadline if migrated. Chang couples this with load sharing, either sender-initiated or receiver initiated. Sender-initiated load sharing provides lower miss rates and lower average lateness at low to moderate utilization, but deteriorates rapidly at high utilizations, particularly if the communications is slow.

Under the global priority system, an attempt is made to migrate tasks so as to service the highest priority ones first on a system-wide basis. This is done by combining receiver-initiated and sender-initiated polling. When a site receives a task that it will not process imediately, due to the presence of higher priority ready tasks, it polls a limited number of randomly chosen sites to see if any of them would give immediate service to the low-priority task; if so, the task is transfered. Conversely, when a site is about to begin servicing a low priority task (because it has no ready high priority tasks), it polls a limited number of randomly chosen sites to see if any of them have a waiting high priority task; if so, such as task is transferred.

Either of these selection heuristics could be combined with our global load balancing scheme, at the point where our scheme is trying to decide which objects to migrate between a particular pair of sites to transfer a particular amount of load. For example, objects could be migrated that have messages waiting that are of higher priority than those on the recipient site, where possible.

3.2.2 Uncertainty and delays

Other authors have also recognized the need of load-distribution systems to employ uncertain, incomplete, and outdated information, and have suggested using Bayesian techniques and/or exploiting historical load data to cope with these conditions. In this section, we will review three of these previous works that are somewhat similar to our own in their focus on such techniques.

Stankovic: Bayesian decision theory

Stankovic [56] provided a general methodology for applying Bayesian decision theory to distributed control problems involving information (such as site loads) that is only available in up to date form locally on each site, but which must be used in decision making on other sites. He presented this methodology in the context of an application to a particularly simple local load balancing problem, but since the methodology itself is considerably more interesting than the particular application, it merits more general description.

Bayesian decision theory provides a criterion for choosing an optimal action given the following data:

1. the *a priori* probability that the system is in each of an enumerated list of states

2. the conditional probability, given each of those states, of making each of a list of possible observations

3. the actual observation which has been made

4. the utility (i.e., desirability) of each action when the system is in each state

In Stankovic's technique, each state is some conjunction of information local to the various sites. For example, in an application to diffusion load balancing, one state might be the neighbor to the North is the least loaded neighbor and at least four tasks less loaded than the decision-making site. Saying that the system is in this state is a statement about the actual instantaneous conditions on all the relevant sites; hence the decision maker is never in a position to know whether the system is in a particular state at the moment or not. However, given suitably synchronized clocks, it is quite possible to find what state the system was in at some past time.

Although the sites don't know each other's current information (load), they have information on what it was when last communicated. Stankovic suggests aggregating this outdated information into observations in the same way current information is partitioned into states. For example, in the diffusion system there would be an observation that meant that the most recent load received from the northern neighor is lower than the most recent loads received from the other neighbors and also at least four tasks less than the local load.

Recall that the actual system state is available, just not until a later time. Therefore, it is possible to occasionally calculate the relative frequency of the various states, and use this as an estimate of their probabilities in all Bayesian decision making until the next time these frequencies are recalculated. Similarly, it is possible to occasionally calculate the frequency with which each observation was made in each state, and take this as an estimate of the conditional probabilities. Stankovic assumes that the utilities of each action in each state are provided by the sytem designer. Given these three ingredients (the estimates of the state probabilities and observation conditional probabilities and the utilitiy function), it is possible at these infrequent moments to calculate a table mapping each observation into the optimal action to take. Then, until the next time that the frequencies are recalculated and the table updated, all that is required is to look up the current observation in the table in order to find out what to do.

This approach is admirable in many regards, but it also suffers from some serious shortcomings. Perhaps the most serious is that in any application where the states summarize information about the entire ensemble, the approach does not scale well to large numbers of sites. Thus, it does not appear directly applicable to our problem of global load balancing.

Another important respect in which Stankovic's scheme is weaker than ours is that it bases each decision only on the most recent available data (using older data only indirectly, for occasionally updating the probability estimates), and it also makes no use of the age of the observations. This will work fine if the observations are of a relatively constant age and that age is such that there is a relationship between states seperated by that delay. However, under other circumstances important information may be thrown away.

For an artificially simple example of how this can occur, consider a system that deterministically cycles between three states, making one state transition every microsecond without fail. Moreover, each observation incorporates only information of a single age; that is, each observation is a fact about what state the system was in at some prior time. Unfortunately, the observations are of ages uniformly distributed over a range of a dozen microseconds. Therefore, the conditional probability of each observation in each state is 1/3. That is, under Stankovic's assumptions, the observations provide no information about the state. Since the prior probabilities of the states are also all 1/3, the decision maker has absolutely no clue what state the system is in. Yet, Stankovic presumed synchronized clocks and time-stamped messages. Thus, in fact any observation allows the current state to be pinned down at least to even odds between two of the states, and frequently even more precisely.

In a periodically loaded system such as we consider, the recent past may not be as good a predictor of the present as older data (from a full period prior). There is a relationship between the load at any prior time and the current load, but that relationship varies. Unlike Stankovic's method, our own approach directly incorporates information of all ages, rather than only the most recent information, treats each piece of information in accordance with its actual age, rather than lumping together all latest observations, and provides optimal weighting to each age of information in accordance to its statistical relationship to the present state.

Chou and Abraham: linear predictive estimation

Chou and Abraham [16] share with Stankovic and us the emphasis on explicitly coping with outdated information. Unlike Stankovic's method, but like ours, their scheme attempts to exploit the time evolution of load and to estimate the current load from multiple observations of past loads, treated differently in accordance with their varying ages. In particular, Chou and Abraham use a so-called "linear predictive" estimation method to allow historical load information to be used to predict present loads. This is similar in spirit to our use of time-series analysis, but differs in several important respects:

- Chou and Abraham used a general adaptive moving-average operator for their prediction. This requires a relatively large number of free parameters be fitted to the load data. Moreover, no choice of the parameters adequately characterizes the load over long time scales, so the parameters are re-fitted each time interval at each site. This fitting requires time proportional to the square of the time window size, that is, the maximum age of historical load information used. Finally, the estimation technique is not based on an underlying model explaining how the load changes; this presumably explains the need for large numbers of free parameters to be continuously adjusted.

In contrast, our algorithm starts with an explicit model of how the load varies that not only is succesful for prediction but also is consistent with an understanding of the underlying mechanisms. This model has only two free parameters that need to be fit, and none the less is sufficiently adaptive to long-term variations in loading that we can do the fitting in advance, using typical sample load histories. The updating that needs to be done each interval at runtime takes constant time (assuming precise historical information, as Chou and Abraham do), and incorates historical information of unbounded age.

- Chou and Abraham used linear prediction separately to predict the load of each site (on each other site), while our method does only a single prediction on each site each time interval, namely of the system-wide average load. This is because Chou and Abraham's version of global load balancing calls for placing each task on the least loaded site, while ours calls only for the movement of work form sites with above-average load to those with below-average load. Chou and Abraham's version clearly wouldn't scale well to large ensembles, and additionally can suffer from instability when all new tasks generated in the entire system get piled on to the single least loaded site, driving it to overloadedness.

- Chou and Abraham assumed that the historical load information used as the basis for the forecast is exact, while we use Bayesian inference to allow estimates of the historical information to be used. This does require weakening our claim to do only constant-time computations at runtime; the Bayesian inference procedure involves computation linear in the time over which the historical information is improved. This time period will typically be similar to the window size in Chou and Abraham's scheme; however, our computation is linear rather than quadratic in this number of time intervals, and is used only to provide a benefit missing from Chou and Abraham's scheme.

Pasquale: decision theory, filtering, and randomization

Pasquale's dissertation [49] combines elements of Stankovic's and Chou and Abraham's approaches, as well as introducing some new elements. Like Stankovic, Pasquale uses Bayesian decision theory to calculate the expected utility of sending a job to each candidate site. (Pasquale's sample implementation is in the context of independent user jobs in a distributed system. Only new jobs may be placed on a remote site, and all sites are candidates.) Like Chou and Abraham, Pasquale uses linearly filtered loads as the basis for his decisions. The unique elements to Pasquale's approach, beyond the specifics such as the details of the linear filtering, are that he uses decision theory also to decide when it is worth communicating load information and that he chooses a decision randomly, weighted by expected utility, rather than always choosing the decision with the highest expected utility. In this section, we will elaborate on each of these points.

Like Stankovic and Chou and Abraham, but unlike ourselves, Pasquale tries to provide each site with information about the load on every other site. Like Chou and Abraham, but unlike Stankovic, Pasquale acts not on the instantaneous loads, but rather on time-averaged (i.e., filtered) versions of those loads. Unlike Chou and Abraham, who disseminate the unfiltered loads and require each site to perform the filtering on the loads from every other site, Pasquale proposes having each site filter its own load and disseminate the filtered version. In fact, he suggests going one step further and disseminating only the result of rounding the filtered load to the nearest multiple of four tasks, with a bit of hysteresis around the midway points so that the rounded value won't wobble back and forth if the filtered value happens to be near the midpoint between two multiples of four.

Pasquale begins his filtering with a first-order autoregressive filter, also known as an exponentially-weighted moving average (EWMA). That is, if we denote the unfiltered load at time t as l_t and the filtered load as f_t, Pasquale lets $f_t = \lambda l_t + (1-\lambda) f_{t-1} = \lambda l_t + (1-\lambda)\lambda l_{t-1} + (1-\lambda)^2 \lambda l_{t-2} + \cdots$, where λ is a parameter between zero and one. Pasquale chose $\lambda = .03533838$ in his experiments, but doesn't report how he made that choice; for that matter, the choice of the first-order autoregressive filter is also unjustified. As it happens, though, EWMA filters of this form are the optimal forecast functions for the class of stochastic processes known as IMA(0,1,1), which we discuss in section 4.3 as the aperiodic analog of our own load model. Thus, although Pasquale doesn't justify this stage of his load filtering process, it is in keeping with our own work on statistical load modelling, as well as the long tradition of using EWMAs for computer system load averages.

However, Pasquale does not stop with this autoregressive filtering (or a rounded version). Instead, he then applies a moving average filter to the result, to attempt to find a "long-term load level" rather than merely the "fundamental component" of the load. In his experimental work, he chooses without justification to use a moving average that gives equal weight to the 60 (already EWMA filtered) load values computed at one second intervals over the past minute.

Pasquale presents a graph of a sample of the actual load, the result of the EWMA filtering, and the result of this additional moving average filtering and the rounding. Judging by that graph, the additional moving average filter does make the ultimate rounded load level more broadly reflective of the *past* minute of activity, but actually hurts the predictive relationship to *future* loads. Since Pasquale provides no formal model for how the load changes over time, he seems to make the mistake of confusing fitting past loads with forecasting future loads (which is what is relevant for decision making). By contrast, our own work is careful to start with an explicit load model and derive from it an optimal forecast function (similar to the EWMA, but exploiting periodicity in our systems' loads).

Although Pasquale's specific illustrative application of his ideas includes these quite *ad hoc* choices of load filtering, his basic design principles are quite similar to ours. In particular, he writes:

> To conquer uncertainty, agents are given knowledge specific to the domain of load dynamics. This knowledge is gathered by observing variables of interest, such as [the load level], and noting specifically how they change *over time*. Most of this knowledge can be acquired offline, applying time-series analysis to past histories of these variables.

This could equally well have been written to describe our own work. Pasquale also emphasizes that this knowledge of how loading changes over time is specific to the particular class of system under consideration, which is other fundamental assumption of our thesis.

Because the filtered loads are coarsely rounded (as well as heavily filtered), and care is taken to apply hysteresis in the rounding, the rounded load levels do not change frequently, and hence do not need frequent communication. Pasquale goes beyond this, though, in imposing what he calls "frugal communications." The sites use Bayesian decision theory to compare the expected utility loss from continuing to use an outdated load value with that caused by the communication costs of updating the load value. Only if the updated load value will make enough of a difference in decision making to pay for the communication is the new load value sent. This is an important general idea, which unfortunately does not seem to fit well into our own approach to estimating the system-wide average load in a distributed manner.

A final interesting contribution of Pasquale's, more similar to our own work than to that of Stankovic or that of Chou and Abraham, is his use of randomization. This is intended to address a problem present in the other related approaches we've described. Stankovic and Chou and Abraham do not provide any mechanism to prevent all sites from choosing the same site (the one appearing least loaded) as the recipient for their work. This can cause the kind of system instability illustrated by a quip of Perlis, "It's so crowded that no one goes over there," quoted by Huberman and Hogg [33].

To prevent this, Pasquale has each decision maker randomly choose among the options for job transfer destinations, including also the option of deferring the transfer. The probability of deferring the transfer is set using estimates of number of jobs that all the sites are trying to place and the total capacity of the underloaded sites for new jobs. The probibilities of the remaining choices are set in proportion to their expected utilities. The goal of this process is to ensure that the total number of jobs transferred will not greatly exceed the available capacity and that they won't all be transferred to the same site.

In our own work, we have used a randomized choice of communications partners and times to assure that not all overloaded sites find the same underloaded site (or vice versa). We also prevent excess work transfer by adopting the global load balancing standard. That is, work is only transferred from an overloaded site to an underloaded, and only enough to bring whichever of them is closer to the system-wide average load to that average. Finally, we provide a "reservation" mechanism to prevent two overloaded sites from simultaneously attempting to fill the same underload on a single underloaded site, even if the randomization should happen to allow both overloaded sites to discover the same underloaded site simultaneously.

3.2.3 Information dissemination

In addition to the techniques for coping with delayed and uncertain information, another fundamental mechanism used in this thesis is the randomized spreading of load information. Other authors have reported similar randomized techniques for disseminating information, as well as non-randomized techniques with similar goals.

Drezner and Barak [22, 23] proposed and analyzed a randomized information dissemination mechanism similar to ours and applied it to load averaging [6]. In this algorithm, each site sends its estimate of the average to one other randomly chosen site each time interval. Our own algorithm generalizes this by sending the estimate to a randomly chosen sample of sites of some constant size. In section 4.2, where we analyze our algorithm, we show that the original version (where the sample size is one) is only optimal under certain rather narrow circumstances that don't include our target system.

Barak and Kornatzky [7] described these same two algorithms in the context of general design principles which motivate them, as well as providing an extension to the first algorithm that allows each site to obtain information from each of a random sample of other sites of predetermined expected size. This technique, while potentially useful for local load balancing or load sharing, does not appear useful for our goal of global load balancing.

Barak and Shiloh [8] implemented a similar load information dissemination mechanism in the MOS distributed system; it provided information on the individual loads of a sample of sites rather than estimating the average load. In order for this algorithm to scale well in cost as the ensemble is increased in size, it is necessary that the sample size remain fixed at a comparatively small value. As with Barak and

Kornatzky's technique for randomized sampling, this could serve local load balancing or load sharing, but not global load balancing.

Alon, Barak, and Manber [2] analyzed techniques similar to those of Barak and Drezner but using deterministic communication patterns in place of randomization, while retaining some degree of robustness (if any site stops participating, information from any other site will eventually make its way to all other sites). These patterns again presume that each site sends to only one other site per round, and so are likely also suboptimal under our circumstances. Further, we favor the randomized version because it also provides a stable basis for matchmaking between overloaded and underloaded sites and because it is comparatively unlikely to cause systematic patterns of network congestion.

3.2.4 Migration and communications redirection

One class of underlying mechanisms that this thesis takes largely for granted is the ability to move objects and their associated tasks from one processing element to another, including in particular the ability to redirect all messages for that object to the new site. One reason why this is treated so lightly here is that it has been quite extensively researched by others, including at least three other Ph.D. theses.

Fowler's thesis [28] provides amortized analysis of upper and lower bounds and estimates of average case costs for several decentralized message redirection protocols based on forwarding addresses. In particular, he shows the benefits gained from compressing forwarding paths when they are used, by modifying each forwarding address accessed in the course of delivering a particular message to point directly to the site to which the message was ultimately delivered. This is the technique which we use in our experimental work.

Artsy, Chang, and Finkel [4] described a distributed operating system, Charlotte, in which process migration was actually implemented. In Charlotte, no information (including forwarding addresses) is left behind when a process is migrated. Instead, all communication partners are notified of the new location before the migration is completed. This is practical in part because Charlotte communication links can only have a single process on each end, so it is cheap to maintain a "back pointer" to the process on the other end of the link. (In fact, Charlotte links are two-way communications channels, so there is no distinction between forward pointers and back pointers.)

In our LAMINA system, by contrast, an object may receive data on a single stream from many potential senders, only some subset of which may actually send anything in the near future. Thus, it would be expensive to maintain a complete set of back pointers and to update all of the senders. (This was discovered empirically in preliminary experiments where this scheme was used.) Doing so in a synchronous fashion (as in Charlotte) is particularly unattractive in a large-scale real-time system such as ours, since it could substantially delay migrations.

Ravi's thesis [52, 51] goes beyond Fowler's primarily in that it attempts to prevent long forwarding chains even for the first message sent by a particular sender after the recipient has migrated. This is done by preemptively multicasting the new location to progressively wider network neighborhoods after progressively longer delays. Thus nearby sites have quite up-to-date location information, while those further off have less up-to-date information. Ravi showed this to provide good performance by using three assumptions, none of which are true in the systems considered in this thesis:

1. migrations are random walks within the network topology, hence spatially local

2. communication frequency is related to spatial distance

3. communication latency is proportional to distance in the network

It is because none of these apply that we have not adopted Ravi's technique.

Jul's thesis [36] provides a complete design of a system for fine-grained object migration in a local area network of workstations, including not only object finding based on Fowler's work but also the encoding of the objects and tasks for transmission and decoding upon reception and a distributed garbage collection algorithm. Jul's encoding technique for objects relies on descriptors to find all embedded pointers, while our simulation presumes tagging support for this purpose instead. Jul's work provides a good example of the engineering needed to make object mobility a practical reality. The garbage collection algorithm, for example, might merit consideration in an actual implementation of LAMINA on a real ensemble machine. For the experiments described in this thesis, however, we used a simulator that doesn't model garbage collection activity.

Totty [58] developed for his Bachelor's thesis an operating-system kernel supporting object migration on the Jellybean machine, a modern ensemble architecture rather similar to our own CARE. He used a forwarding address scheme with the forwarding addresses stored in limited-size associative translation buffers. When it proves necessary to find an object whose address was displaced from the local translation buffer, the object's "birth site" (which is included in the object identifier) is used as a fall-back guaranteed to have an address for the object. This trades frugality of memory consumption for potential bottlenecks at the birth site. In our simulation, we chose instead to presume that ample memory was available at all sites, in part because that reduces the number of anomalous performance effects that can complicate the interpretation of our experimental results.

3.3 Summary of related work

Research contributions in the general area of load distribution are distinguished not only by the problem-solving methods employed, but also more fundamentally by the problem attacked. For example, any method for mapping a structured data domain onto the network structure at compile time is fundamentally distinct from any method for independently siting individual objects at runtime. Thus, the first section of this chapter is best seen as illustrating the degree of similarity between our own problem (global load balancing migration) and other load distribution problems, rather than focusing on the techniques used.

Although this taxonomic view serves to delimit the problem area in which our contributions fall, it has the unfortunate effect of comparing our work to dissimilar prior work rather than similar. To remedy this, the second section of this chapter highlighted particular areas of similarity between our work and prior work. The similarity to our own work grew progressively stronger as we moved from each of the four areas to the next. There is comparatively little in common between our work and other real-time system work; instead, our work is more closely aligned with other attempts to use outdated load information. Even in this latter area, our techniques are novel. By contrast, our information dissemination algorithm directly generalizes and improves upon a prior algorithm, and in the area of object migration and communication redirection this thesis offers nothing new beyond concrete implementation choices.

This background serves to set the stage for the following chapters. We shall present a new approach to using uncertain historical load information and a generalization of Drezner and Barak's randomized averaging algorithm in the next chapter, while leaving the migration mechanics for the chapter on implementation issues.

Chapter 4

Modeling and Algorithms

Our load-balancing algorithm consists conceptually of the following components; in actuality, some of the components are integrated, as will be seen later.

- A load metric is used locally on each processing element to quantify the load on that processing element's evaluator (main processor).
- A distributed averaging algorithm estimates past system-wide average loads; the estimates improve in accuracy with time.
- The time-series forecast uses those estimated average loads for past times to estimate the current average load.
- A partner-seeking mechanism allows overloaded and underloaded sites to find each other.
- A policy determines which objects to migrate in order to shift a given amount of load between a given pair of sites.
- The actual migration mechanism relocates the arbitrarily graph-structured state of each migrated object and updates addresses used by other objects to communicate with the migrated objects.

The migration mechanism will be introduced in the next chapter in the context of the actual implementation; the other components are described in the remainder of this chapter.

4.1 Load metric

One of the first questions confronting the designer of a load balancing system is "What is load?" If the load is to be balanced, imbalances must be identified, and that requires quantifying the load on each site.

The first question is which system resources should have their loading balanced. It is perfectly plausible to try to balance the use of the various communication channels, message routers, or housekeeping processors, or to balance the amount of memory used at each site for storing objects' state. However, this thesis focuses instead on balancing only the use of the evaluators, the main processor at each site. This is because previous experience with the system identified this as particularly problematic, and because evaluator load imbalances seemed to be the underlying cause of many other imbalances. For example, communications congestion was also limiting system performance, but generally was caused by the heavy traffic to and from an especially busy evaluator. Similarly, the housekeeping processors (operators), which were typically lightly utilized, often were especially busy processing context switches and message receptions and transmissions on precisely those sites where the evaluator was very busy.

Having chosen to target the evaluator processing load, it is still necessary to choose a particular metric of that load. Utilization (i.e. the percentage of time busy) is a poor metric, because it saturates when the processor is fully utilized, failing to differentiate between a processor that has just enough work to keep it busy and one which is swamped with a long queue of processes waiting to run. Therefore, we are lead to the (conventional) choice of measuring the size of the queue of waiting processes. In our object-oriented system, this amounts to measuring the queue of messages waiting for delivery, aggregated over all the objects located on the site.

The ideal unit of measure for these message queues would be the nanosecond—if the processing time needed to respond to each message were known, a sum over the pending messages of those processing times would provide an accurate picture of the work available on that site. In the system under consideration, however, the processing times are not available *a priori*; since they tend to be relatively uniform, however, a simple count of pending messages serves as an adequate load metric. This is what was in fact used for the experiments described in this thesis. At the expense of greater overhead, a more precise estimate could be obtained by checking the message type and recipient class for each queued message and looking up in a table the average execution time for that method. One reason this wasn't used in the experiments described here is that it would have produced unrealistically perfect estimates of the execution times, since as reported in section 2.4, the same technique was used to provide the execution times for the simulation.

4.2 Distributed averaging

A primary contribution of this thesis is the use of an explicit time-series model of loading to allow sites to estimate the current system-wide average load when all they can obtain information about is past average loads. This is essential, because in an ensemble architecture there can never be up-to-date global information available. In the following sections we will see how this is done; for now, we turn our attention to how the historical data is collected that serves as the foundation for that process.

Because this data is used together with the time-series model, the method of determining past average loads can be as slow as necessary to fit within a constrained resource-utilization budget, even if that would be too slow to be used directly without forecasting. Obviously, however, the faster it is, the less extrapolation the time-series model will be forced to do, and consequently the better we can expect our results to be. In this section we optimize the speed of a particular distributed average-determination process, subject to constrained resources.

This section also develops an explicit model of how the historical information improves in accuracy with age. In section 4.4 we will show how this model can be used together with that for the loading to allow optimal integration of the data of varying age (hence relevance) and accuracy. This is another major contribution of the thesis.

One strategic decision is the general style of distributed averaging algorithm. This thesis focuses on a randomized algorithm that ignores the locality of processing elements in the interconnection network. This choice allows even the earliest estimates, incorporating data from only a few processing elements, to be (statistically) representative of the whole system, rather than of a limited neighborhood. This can be important if the loading of neighboring processing elements is correlated, as is often the case. Because the target architecture incorporates cut-through routing, the lack of communications locality is not as major an obstacle to efficiency as it otherwise would be. Finally, this randomized, non-local averaging process integrates well with the partner-seeking component to allow non-local object migrations, a fundamental objective of our design.

Our average-determination process is a variation on one proposed by Barak and Drezner [6]. The following description shows the estimation of the average load for only a single time; the actual implementation needs to essentially run one copy of this algorithm for each time interval. (We assume that the time period of the input has been divided into some integer number of intervals for load estimation.) The n sites begin at some time t with their own load as an estimate of the average at that time. They then go through a number of rounds in which they improve their estimates of the average load at time t. In our implementation, the rounds of improvement correspond with the time intervals used for load measurement, though in principle either could be an arbitrary multiple of the other. Note that the estimates are being increased not only in accuracy but also in age; seen in absolute time the improved estimates at the end of each interval are still of the average at time t, but seen in relative time they have changed from being estimates of the average ΔT intervals ago to being estimates of the average $\Delta T + 1$ intervals ago.

In each interval each site multicasts its current estimate to some fixed number (m) of randomly chosen sites. At the end of each interval each site replaces its current estimate with the average of that estimate and the estimates received during the interval. In Barak and Drezner's version of the algorithm, $m = 1$. We investigate below alternative values for this parameter.

Provided that there is a large number of processing elements, Barak and Drezner showed that the variance of the estimates about the true average decreases geometrically with the number of rounds. The analysis below shows that the rate of decrease per round is greater for larger values of m. This suggests that we should choose a large value of m to speedily arrive at sufficiently good estimates. On the other hand, each site will be interrupted $m+1$ times in each interval on the average (once by an interval timer and an average of m times by incoming messages), so a high value of m may necessitate a longer interval and actually result in a *slower* convergence on the true average.

We will show how a balance can be struck between these considerations. Our goal is to choose m so as to optimize the rate of convergence (measured relative to time, not intervals) given a fixed limit on the rate at which the averaging process may utilize system resources. For illustration we will take a simplified estimate of the overhead of context switching, message transmission, and message reception to be the limited resource. A more careful accounting of actual operator processing and operator/network bandwidth gives the same result.

In contemporary systems, it is expected that context switching and message overheads at the sender and receiver will be the limiting resource—rather than raw communications bandwidth, for example [3, 18, 10]. (The actual computation involved in averaging together a few estimates is not likely to dominate.) Message transmission and reception overheads as well as context-switching overheads are approximately proportional to $m+1$ in our system because it provides hardware support for multicast [13]. (Thus there is one transmission and m receptions, rather than m of each.) We therefore calculate the choice of m which will optimize the rate of convergence of the average estimates, subject to a fixed interrupt rate.

Barak and Drezner showed in general that the variance decreases in each round by a factor of $\left(\sum_{k=0}^{n} \frac{p_k}{k+1}\right)^{-1}$, where p_k is the probability that a site receives estimates from k other sites in an interval. Since in our case each of the n sites sends its estimate to m randomly chosen from the n, the number of estimates received will be binomially distributed, with $p_k = \binom{n}{k}(m/n)^k(1-m/n)^{n-k}$.

Plugging this in, and using a mathematical trick borrowed from Barak and Drezner's analysis, we see that the rate of convergence is

$$\begin{aligned}\left(\sum_{k=0}^{n}\frac{\binom{n}{k}(m/n)^k(1-m/n)^{n-k}}{k+1}\right)^{-1} &= \left(\frac{n}{m}\int_0^{m/n}(x+1-\frac{m}{n})^n\,dx\right)^{-1}\\ &= \left(\frac{n}{m(n+1)}(1-(1-\frac{m}{n})^{n+1})\right)^{-1}.\end{aligned}$$

This is approximately $m/(1-e^{-m})$ when n is large. (Note that n doesn't need to be especially large for this approximation to be a good one. For example, for $m = 3$, even at $n = 64$ the error is less than one percent.)

The number of intervals needed to achieve a particular degree of convergence is inversely proportional to the logarithm of this number, while the length of each interval must be a fixed multiple of $m+1$, according to our model of what constitutes fixed resource consumption. Since the time needed to achieve this fixed degree of convergence is the product of the number of intervals and the length of each of those intervals, the time must be some proportionality constant times $\frac{m+1}{\log m - \log(1-e^{-m})}$. Therefore, for best performance we want to find the value of m which minimizes that fraction. Constraining m to be integral, we find that the optimum is three. It might be possible in principle to drop the requirement that m be an integer, e.g. by having each site make a randomized choice of multicast breadth each interval. However, examining the graph of this function in figure 4.1 we see that three is close enough to the true optimum that this doesn't appear worthwhile.

This analysis allows us to conclude that if interrupt handling limits the rate at which averaging actions can be performed, each site should send to three others in each interval, rather than one as Barak and Drezner suggest, to optimize the speed of convergence of the average estimates. The next question is, how sensitive is this conclusion to the assumption that the interval length would have to be proportional to $m+1$ in order to stay within a fixed resource budget? In particular, we will show that the choice of $m = 3$ is reasonable given a more complete model of costs in CARE, and characterize the circumstances under which Barak and Drezner's choice of $m = 1$ makes sense.

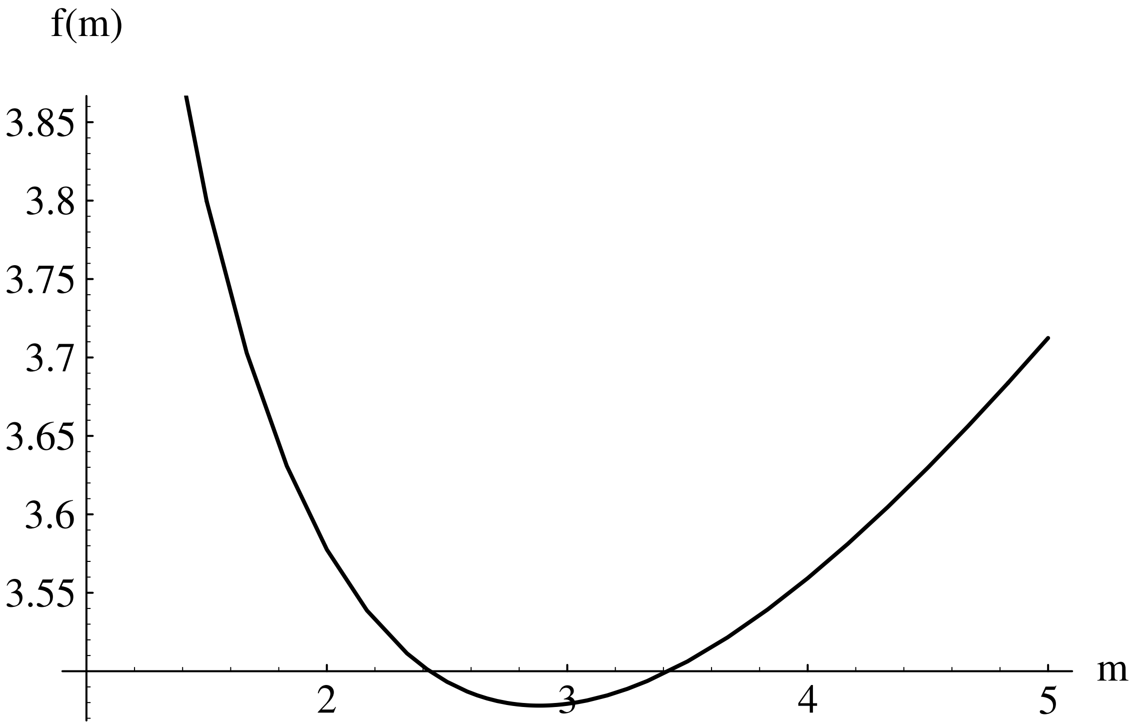

Figure 4.1: Averaging time vs. m (cost $\propto$ $m+1$). Assuming that the resource utilization per interval is proportional to $m+1$, this graph of $f(m) = \frac{m+1}{\log m - \log(1-e^{-m})}$ shows that the optimum value for m is near 3.

In any implementation of our algorithm, there will be some work that needs to be done each interval independent of m, while other costs will scale proportionately to m. Thus, the total cost of each interval will be some linear function of m, i.e. $Am + B$ for some constants A and B that reflect the hardware architecture, software implementation, and choice of cost model (i.e., which resource constrains the interval: network to operator interface bandwidth, operator processing time, total network bandwidth, ...). The foregoing analysis, which assumes that the cost of each interval is proportional to $m+1$, applies to the case where $A = B$, i.e. where the fixed overhead cost of each interval is equal to the extra cost incurred per target of the multicast.

For the more general case, where $A \neq B$, we need to find the integer m which minimizes some member of the parameterized family of functions $f_\alpha(m) = \frac{m+\alpha}{\log m - \log(1-e^{-m})}$, where the parameter α equals B/A. One interesting limiting case is $\alpha = 0$, i.e. all cost is directly related to m, with no fixed per-interval cost. Taken at face value this seems implausible, but it accurately models a system in which the rate of load averaging activity is limited by operator to network interface bandwidth and there is no network support for multicast (unlike CARE). This change, from overhead being 100% of the per-target cost to it being 0%, is a dramatic change, since we are considering small values of m. For example, it means that the per-interval cost with $m = 3$ is now three times as high as for $m = 1$, while with our previous cost model $m = 3$ would only be twice as expensive as $m = 1$. Therefore, it should come as no surprise that the optimization problem is qualitatively quite different, as shown in figure 4.2. Since this function is monotonically increasing, the optimum choice is $m = 1$, i.e. Barak and Drezner's original proposal. Filling in the gap in the family of curves between $\alpha = 0$ and $\alpha = 1$, figure 4.3 shows the functions for $\alpha = 0$, .2, .4, .6, and 1. As α increases, the function becomes more "hooked" and the optimum value of m increases. If an implementation had particularly high overhead costs, there is nothing to stop α from being even larger than 1; as can be seen in figure 4.4 (which shows $\alpha = 1$, 1.2, 1.4, 1.6, 1.8, and 2), the same general trend in the functions' behavior continues.

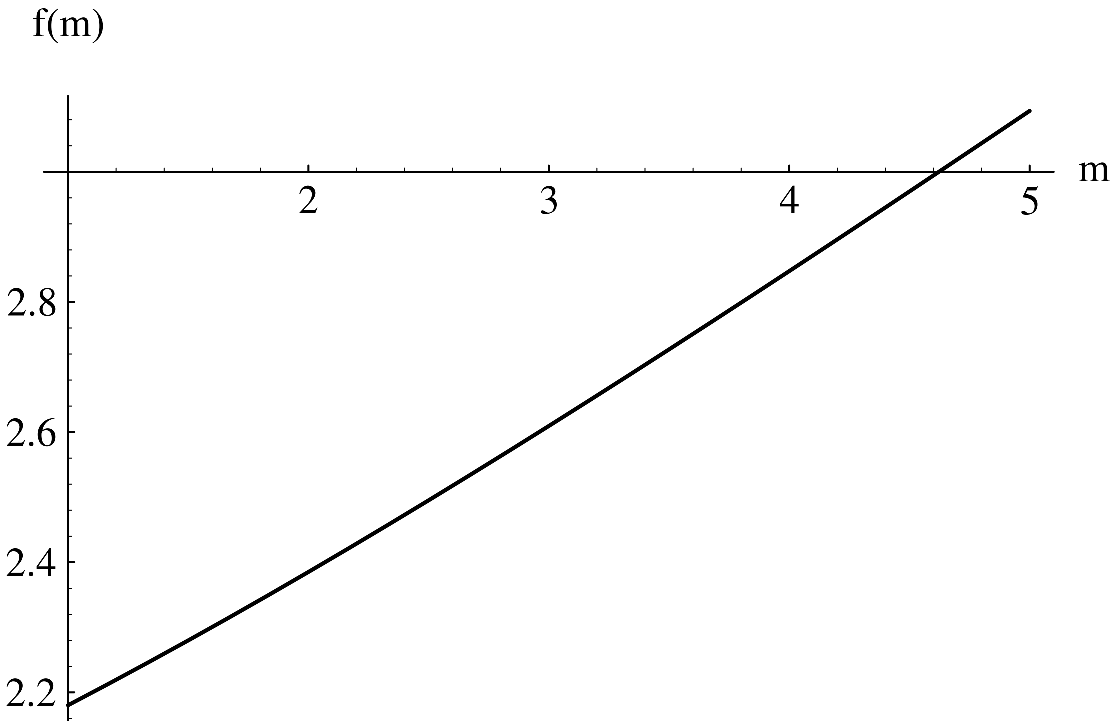

Figure 4.2: Averaging time vs. m (cost $\propto m$). Assuming that the resource utilization per interval is proportional to m results in a qualitatively different optimization problem. This graph shows $f(m) = \frac{m}{\log m - \log(1-e^{-m})}$.

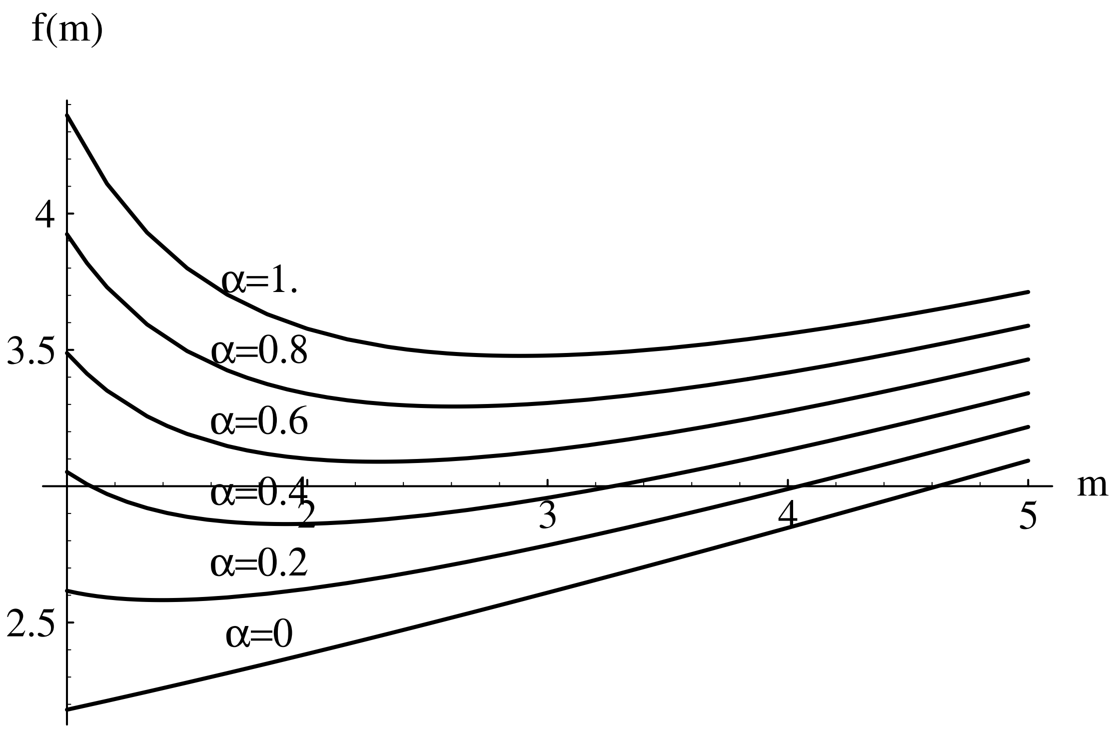

Figure 4.3: Averaging time vs. m $(0 \leq \alpha \leq 1)$. This graph shows members of the family $f_\alpha(m) = \frac{m+\alpha}{\log m - \log(1-e^{-m})}$. As α increases, so does the optimal value of m.

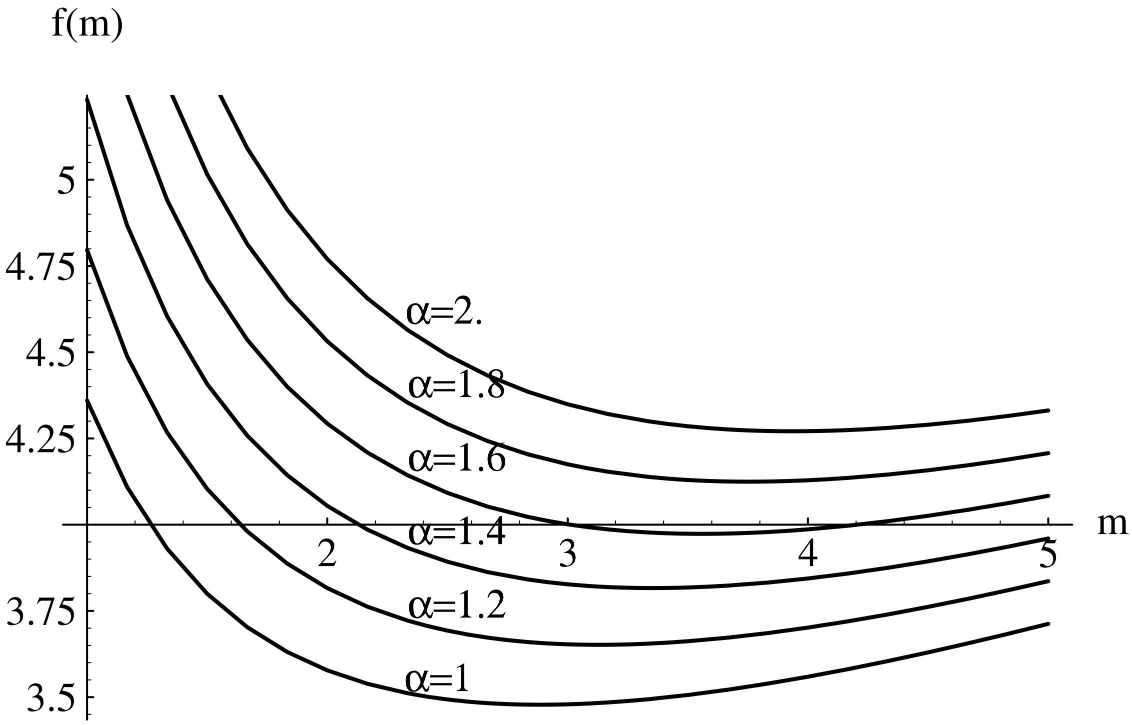

Figure 4.4: Averaging time vs. m ($1 \leq \alpha \leq 2$). This graph shows further members of the family $f_\alpha(m) = \frac{m+\alpha}{\log m - \log(1-e^{-m})}$, for larger values of α.

If we make a table of how the optimal integer value for m depends on α, we get

optimal m	α
1	$0 \leq \alpha \leq .207$
2	$.207 \leq \alpha \leq .695$
3	$.695 \leq \alpha \leq 1.51$
4	$1.51 \leq \alpha \leq 2.64$

(these threshold values for α are rounded to three places). From this, we can conclude that Barak and Drezner's choice of $m = 1$ is only optimal under very limited circumstances, namely when the marginal cost of an extra multicast target is at least five times as high as the fixed cost of each averaging interval. (Note that it is a bit unfair to refer to Barak and Drezner's *choice* of $m = 1$, since their algorithm didn't have the parameter m, i.e. didn't even open up the option of larger values.) Another conclusion we can reach is that our proposal of $m = 3$ is comparatively insensitive to the details of the cost model, in that a modestly large region around our original $\alpha = 1$ still makes $m = 3$ optimal. Even if the per-interval overhead is as little as 70% of the per-target cost or as high as 150% of that cost, $m = 3$ will still outperform other choices.

For the actual CARE implementation described in the next chapter, the number of operator processing cycles spent on the distributed averaging algorithm each interval is $540m + 417$, according to the simulation cost model. Thus if we take operator processing as the limited resource in CARE, we have $\alpha \approx .8$, and are therefore correct in choosing $m = 3$. If instead we take the bandwidth of the operator to network interfaces to be the resource which limits averaging activity, then the same simulation cost model for the CARE implementation yields $\alpha \approx .9$, so also within the region where $m = 3$ is optimal. (It might appear at first that this latter value of α would necessarily be exactly 1, since each message is sent once and received m times, as earlier in this section. However, this fails to account for the fact that the sender has to also provide m destination addresses to the network.)

In summary, we can conclude that if we allow only a fixed fraction of the CARE operators' capacity to be spent on load averaging (accounting not only for interrupt handling and message transmission and reception, but also the actual averaging work and other overheads), each site should send to three others in each interval, rather than one as Barak and Drezner suggest, to optimize the speed of convergence of the average estimates. We have also along the way modeled how the estimates improve in accuracy with age, which will be important in section 4.4.

4.3 ARIMA load modeling and forecasting

The preceding section provides a method of obtaining information about past system-wide loading; in this section we will show how the evolution of the load over time can be modeled, which will allow that historical information to be put to use in forecasting the current load. This model is also important because as we have seen, we can quantify the degree to which older estimates are more accurate. Thus we are now challenged to quantify the counterbalancing force, namely the degree to which those older estimates are (in general) less relevant. The following section synthesizes these two models to optimally weight the data of each age; for now, we will stick to modeling how the load varies with time. This load modeling is in itself a major contribution of the thesis.

Note that although in the long term older loads are less relevant, this is not always the case in the short term. In particular, because the systems studied in this thesis exhibit statistically periodic loads, load information from a full period ago may be more relevant than recent load information. We are still faced with the problem of integrating together data of varying accuracy and relevance, where the weight to assign to each datum must reflect these two counteracting considerations. We have greater hope with a periodic load that our resulting estimate will be usefully precise than with an aperiodic load. This is because we have data available for which both the accuracy and the relevance are reasonably high.

We have chosen to use the general family of stochastic process models known as "ARIMA," i.e., autoregressive-integrated moving average process models. This choice of model is often called "Box-Jenkins analysis" because it was popularized by Box and Jenkins [11]. These models are attractive because highly parsimonious models (i.e., models with few parameters requiring empirical fitting) have proven adequate for many forecasting and control applications. In particular, the subclass of ARIMA models we will use (so-called multiplicative IMA models, which will be explained below) have been successfully applied to the parsimonious modeling of seasonal time series for forecasting purposes [11, Chapter 9]. The statistically periodic nature of seasonal data (sales, for example) is similar to that of the system loading in our soft-real-time systems. In both cases, the behavior in each period is similar to that in nearby periods, but there is no exact repetition from period to period and in fact the overall pattern may gradually shift so that far apart periods bear little resemblance. Further, these models provide adaptive modeling of intra-period as well as inter-period patterns.

An ARIMA model of a time-series characterizes it by a linear filter which transforms a white-noise series (a series of independent normal deviates) into the modeled series. Specifically, the filter is decomposed into a composition of stationary autoregressive and moving-average filters with summing (integration) stages to model any non-stationarity. (A moving-average filter's output is some linear function of the current input and some finite number of previous inputs. An autoregressive filter, on the other hand, has as its output some linear combination of the current input with some finite number of previous *outputs*. A stationary series, roughly speaking, is one that has a well-defined long-term average.) A single summation produces a series non-stationary in level but otherwise homogeneous; double summation allows the slope as well as the level to shift, etc. For periodic series, these concepts can be applied both to the variation between consecutive measurements and to the variation between corresponding measurements in consecutive periods.

Box and Jenkins recommend identifying a potential model for an observed series by the following steps:

1. Perform sufficient differencing on the series to render it stationary, as evidenced by the autocorrelations.

2. Examine the autocorrelations of the resulting stationary sequence, looking for the characteristic pattern of one of the simple ARMA models. (Note that we are

left with only an ARMA modeling problem, not a full ARIMA one, because the differencing in the preceding step corresponded to the summation component of the model.)

3. Form a preliminary estimate of the model parameters by solving for them in equations between the observed autocorrelations and those predicted by the model.

4. Use the sum-of-squares surface to refine these preliminary parameter estimates to approximate least-squares estimates.

5. Use diagnostic tests on the "residuals" (differences between the model's predictions and actual data) to test the adequacy of the model (for a perfect model, the residuals should be white noise).

We have performed this modeling for two different soft-real-time systems: the ELINT system used as the basis for our load balancing experiments and the AIRTRAC-DA system [45]. The latter was not kept up to date with changes in the underlying system software, to the point of no longer being executable, so we were unable to perform load balancing tests on it. None the less, the successful modeling of its load described in [30] provides some evidence of the more general applicability of our approach.

In both cases, the modeling procedure described above leads us to a multiplicative IMA model of orders $(0, 1, 1) \times (0, 1, 1)_p$, where p is the number of load measurements per input period of the system. What this means is that the overall model separates neatly into the composition of an intra-period model with an inter-period model, and that both components combine a "zeroth-order" autoregressive operator (i.e., no autoregressive operator), a first-order moving average operator, and a single summation. (It is because there is no autoregressive component that this is called an IMA model, rather than ARIMA.) If we denote the white noise series by a_t, the load series by z_t, and the parameters of the two moving average operators by θ and Θ, the model can be written as $z_t = z_{t-1} + z_{t-p} - z_{t-p-1} + a_t - \theta a_{t-1} - \Theta a_{t-p} + \theta\Theta a_{t-p-1}$. The parameters θ and Θ must each be less that one in absolute value; the variance of the white noise series a_t is also a parameter, which we'll call σ_a^2. In the above equation, the z_{t-1} and z_{t-p} terms come from the intra-period and inter-period summations, respectively, with the $-z_{t-p-1}$ term being from their interaction. Similarly, the $-\theta a_{t-1}$ and $-\Theta a_{t-p}$ are from the two first-order moving average operators, with the $\theta\Theta a_{t-p-1}$ being their interaction.

Intuitively this model makes sense. Looking at the aperiodic component, what we are saying is that the load on the system at any instant is the same as it was the previous instant plus some perturbation (hence the summation operator). The perturbation is not independent from moment to moment, however. Some fraction determined by θ of each perturbation is carried over to the next interval (hence the moving average operator). That this is a reasonable model for the ups and downs of processing load is evidenced by the fact that its optimal forecast function, the exponentially weighted moving average, is the standard in computer system load estimation. The periodic component has the same form, and essentially states that the longer term shifts in loading caused by entries and exits of aircraft from the observation area, e.g., follows the same pattern as the short term shifts caused by the activation and completion of processing tasks.

4.4 Time-series forecasting from average estimates

The time-series model of section 4.3 would directly show how to forecast the current system-wide average load, if what we had was precise observations of the average load up until some previous time, and no observations thereafter. However, what the distributed averaging algorithm of section 4.2 actually provides as raw material is observed estimates of the average, with accuracy geometrically increasing with age. Thus the information we have combines two sources of uncertainty:

1. the uncertain connection between the estimated past loads and the actual past loads, and

2. the uncertain connection between past loads and the current load.

The two models from the preceding sections directly address these two sources of uncertainty; one describes the accuracy of the various historical estimates, and the other describes the autocorrelation structure over time of the load. The goal of this section is to integrate together these two models so as to produce the best possible estimate of the current load. We do this using the technique of Bayesian inference; the following subsection serves as an introduction to the concept of Bayesian inference, using a simplified version of the problem. We then move on to generalize that univariate solution to a multivariate version that meets our actual needs.

4.4.1 Bayesian inference

To clarify the situation, consider for the moment a simplified version in which the distributed averaging algorithm provides exact average loads for all past time intervals and estimates with a known uncertainty for the current time. In what follows, we will call these estimates provided by the averaging algorithm "observations" rather than estimates, so as to reserve the latter term for estimates based at least in part on the time-series model. To be more specific, assume that the observations come from a normally distributed process, with the true current average load as mean and a known variance; we will call this distribution the observation distribution.

Even without the averaging algorithm's observation of the current load, we can estimate what the current load is using the prior loads and the time-series model. In fact, since the time-series model is in terms of an explicit stochastic process, we can not only give our best guess but also quantify our degree of certainty by expressing this "prior" estimate as a probability density function for the current load. Using an IMA model as in section 4.3, this prior distribution will be normal.

Now the averaging algorithm provides an observation, which gives additional information about the current load. Because of our assumption about the averaging algorithm, this observation is a sample drawn from the normal observation distribution, which has the unknown actual current load as its mean and the known accuracy of the estimate as its variance. So, we have a normal prior distribution for the mean of this normal observation distribution, know its variance, and now have one sample drawn from it.

If that sample is a surprising one, i.e. lies far from our prior estimate of the mean, then we will want to revise that estimate of the mean. In fact, we will always revise the estimate unless it exactly agrees with the observation; the only question is one of degree. How much we revise the estimate will depend on how surprising the observation is and how confident we were in our prior estimate. In other words, it depends not only on the difference between the observation and the prior estimate of the mean but also on the relative variances of the two normal distributions.

If the prior distribution has a large variance (i.e. we didn't have much confidence in our prior estimate) and the observation distribution has a small variance (i.e. an observation generated by the process is very likely to be near its mean), then we will largely disregard the prior estimate and put most of our belief in the sample.

Conversely, if the prior distribution has a small variance (i.e. the actual load was expected to fall very near the prior estimate) and the observation distribution has a large variance (i.e. there is little reason to expect a single sample drawn from it to be near its mean), then we will largely disregard the observation and make little adjustment to the prior estimate.

Just as the prior estimate is expressed as a full probability density function, so too we can express our adjusted "posterior" estimate as a probability density function. This posterior distribution incorporates information from both the time-series model and the model of the averaging algorithm's accuracy.

In this example, where both the prior distribution and the observation distribution are normal, it happens that the posterior distribution will also be normal. In the jargon of Bayesian inference, normal distributions are a "conjugate family" for the mean of normal processes. Moreover, the mean and variance of the posterior distribution can easily be calculated from the mean and variance of the prior distribution and the variance of the observation distribution. In the next subsection we show this calculation in matrix terms for a multivariate generalization of this simplified example.

4.4.2 Estimating from uncertain history

We have described above the basic approach to integrating information from history and the time-series model with information from the distributed averaging algorithm, using Bayesian inference. We now have to see how this can be applied when the historical information is itself uncertain, since it comes from the same distributed averaging algorithm.

In order to simplify the formulation of this problem, we will pretend that we do know exactly the average loads up through some "lead time," l intervals ago. Although this is strictly speaking false, the older estimates from the averaging algorithm are quite reliable. Recall that each interval of averaging reduces the variance of these estimates by more than a factor of three. Because of this geometric convergence of the averaging process, it is possible to select a value of l that is both reasonably small and also makes the above fiction sufficiently close to true. We can then run l copies of the averaging algorithm together, so that each interval the average from l intervals before achieves fully-refined status, and the newer averages are each refined one step. (Rather than having l independent copies running, they are actually fused together in our implementation, with each interval a single multicast message containing l values being transmitted by each site.)

The key insight is that with this simplification, we are in a situation completely analogous to that of the previous section, where we had exact knowledge of past loads and a single observation of the present load. It's just that the "present load" we are observing is now the vector of loads for the latest l intervals. This observed vector of estimated average loads, which results from the distributed averaging algorithm, is a sample of a multivariate process. We will approximate this process by a normal one, relying on the central limit theorem and the large number of processing elements as justification for this approximation.

The analysis of section 4.2 gives the covariance matrix of this multivariate normal process. We would like to estimate its mean vector. Our prior belief about the mean, from the time-series model, is also normally distributed, with both the mean and the covariance known. This is convenient, because as in the univariate case, the multivariate normal distributions form a family of conjugate distributions for the mean of a multivariate normal process. That is, our posterior belief about the process's mean will also be normally distributed, and its mean and covariance can readily be calculated from the mean and covariance of the prior and the covariance of the process.

This is a bit confusing, because of how many means and covariances there are. Therefore, let's introduce a bit of notation. We will call the actual, unknown system-wide average loads for the most recent l intervals $\mathbf{z}$. (We will use subscripts in the range $(1-l), \ldots, 0$ with 0 being the current interval.) The distributed averaging process provides us with a sample, which we will call $\mathbf{x}$, drawn from a normal distribution with mean $\mathbf{z}$ and covariance matrix $\mathbf{\Sigma}$. We express our state of knowledge about $\mathbf{z}$ as a probability density function for the random variable $\tilde{\mathbf{z}}$. From the time-series model and data older than l intervals we will calculate a normal prior distribution for $\tilde{\mathbf{z}}$ which will have mean $\dot{\mathbf{z}}$ and covariance matrix $\dot{\mathbf{\Sigma}}$. Lastly, our Bayesian inference will result in a posterior distribution for $\tilde{\mathbf{z}}$ which is also normal, and the mean and covariance of which we will call $\hat{\mathbf{z}}$ and $\widehat{\mathbf{\Sigma}}$. Symbolically,

$$
\begin{aligned}
f_{\tilde{\mathbf{x}}|\tilde{\mathbf{z}}}(\mathbf{x} \mid \mathbf{z}) &= N(\mathbf{z}, \mathbf{\Sigma}) \\
f_{\tilde{\mathbf{z}}}(\mathbf{z}) &= N(\dot{\mathbf{z}}, \dot{\mathbf{\Sigma}}) \\
f_{\tilde{\mathbf{z}}|\tilde{\mathbf{x}}}(\mathbf{z} \mid \mathbf{x}) &= \frac{f_{\tilde{\mathbf{x}}|\tilde{\mathbf{z}}}(\mathbf{x} \mid \mathbf{z}) f_{\tilde{\mathbf{z}}}(\mathbf{z})}{\int_{-\infty}^{\infty} f_{\tilde{\mathbf{x}}|\tilde{\mathbf{z}}}(\mathbf{x} \mid \mathbf{z}) f_{\tilde{\mathbf{z}}}(\mathbf{z}) d\mathbf{z}} \\
&\stackrel{\text{def}}{=} N(\hat{\mathbf{z}}, \widehat{\mathbf{\Sigma}})
\end{aligned}
$$

Solving for $\hat{\mathbf{z}}$ and $\widehat{\mathbf{\Sigma}}$ in the above equation, we get

$$
\begin{aligned}
\widehat{\mathbf{\Sigma}} &= \left(\dot{\mathbf{\Sigma}}^{-1} + \mathbf{\Sigma}^{-1}\right)^{-1} \\
\hat{\mathbf{z}} &= \widehat{\mathbf{\Sigma}}\dot{\mathbf{\Sigma}}^{-1}\dot{\mathbf{z}} + \widehat{\mathbf{\Sigma}}\mathbf{\Sigma}^{-1}\mathbf{x} \\
&= \dot{\mathbf{z}} + \widehat{\mathbf{\Sigma}}\mathbf{\Sigma}^{-1}(\mathbf{x} - \dot{\mathbf{z}}).
\end{aligned}
$$

Note that this is actually more than we need—all we care about is the value of $\hat{z}_0$, our best guess as to the current system-wide average load. Therefore, while it is convenient to initially formulate the problem as if we were calculating the entire vector, we will eventually arrange to only calculate this one element. We will also examine $\widehat{\Sigma}_{0,0}$, the variance of our posterior estimate.

In order to be more concrete, we will have to specify $\dot{\mathbf{\Sigma}}$ and $\mathbf{\Sigma}$. We can readily derive $\dot{\mathbf{\Sigma}}$ from the IMA $(0,1,1) \times (0,1,1)_p$ time-series model. The matrix will have an especially simple form, which we show below, if we make the simple, but realistic, assumption that $l \leq p$ (i.e., we have sufficiently accurate data about the load one period ago). Although we will use this simple case for illustration, nothing in our analysis relies on the form of $\dot{\mathbf{\Sigma}}$, so the more general case is susceptible to the same analysis. Letting $\lambda = 1 - \theta$, from [11, equations (9.2.11) and (A5.1.3)] we have

$$\dot{\mathbf{\Sigma}} = \sigma_a^2 \begin{pmatrix} 1 & \lambda & \lambda & \lambda & \cdots & \lambda \\ \lambda & 1+\lambda^2 & \lambda+\lambda^2 & \lambda+\lambda^2 & \cdots & \lambda+\lambda^2 \\ \lambda & \lambda+\lambda^2 & 1+2\lambda^2 & \lambda+2\lambda^2 & \cdots & \lambda+2\lambda^2 \\ \lambda & \lambda+\lambda^2 & \lambda+2\lambda^2 & 1+3\lambda^2 & \cdots & \lambda+3\lambda^2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda & \lambda+\lambda^2 & \lambda+2\lambda^2 & \lambda+3\lambda^2 & \cdots & 1+(l-1)\lambda^2 \end{pmatrix}$$

The analysis of section 4.2 allows us to determine the diagonal entries of $\mathbf{\Sigma}$, i.e., the variances of the observations of each age from the distributed averaging process, at least in terms of the variance of the sites' loads. *A priori* it appears that the off-diagonal terms should be zero, i.e. the averaging errors for different times should be uncorrelated. Actually, this is a simplification of the actual state of affairs, since our implementation combines together the averaging algorithms for the various times, and hence they share common sub-trees of processing elements. However, we will proceed with zero terms except on the diagonal, on the assumption that this is not fundamentally flawed, and with the knowledge that the same general approach could be applied to a more careful analysis, since once again we are not exploiting the structure of the matrix.

If we assume that the perturbations a_t in the time-series model of the system-wide load are the sums of independent and identically distributed perturbations at each site, then the loads of the n sites at any particular time are normally distributed with variance $n\sigma_a^2$. This is also, therefore, the variance of an age-zero (completely unrefined) average from the distributed averaging algorithm. The variances of older averages follow directly from this and the analysis of section 4.2.

As one additional complication, we will in general be somewhere in the middle of an averaging interval. Therefore, we will have some number $k \geq 1$ of averages of each age. If we average those together to get our observed vector $\mathbf{x}$, it will further decrease the variances by a factor of $1/k$. Putting this all together, and letting $\beta = (1 - e^{-m})/m$, we get

$$\mathbf{\Sigma} = \sigma_a^2 \frac{n}{k} \begin{pmatrix} \beta^{l-1} & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \beta^2 & 0 & 0 \\ 0 & \cdots & 0 & \beta & 0 \\ 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

One thing to note amid all these greek letters is that because both $\mathbf{\Sigma}$ and $\dot{\mathbf{\Sigma}}$ have factors of σ_a^2, so will $\widehat{\mathbf{\Sigma}}$. This will cancel out against the factor of $1/\sigma_a^2$ in $\mathbf{\Sigma}^{-1}$ when we calculate $\hat{\mathbf{z}}$. Therefore, we can continue to not care what σ_a^2 is, and calculate as though it were 1. (Of course, the variance $\widehat{\Sigma}_{0,0}$ will have a factor of σ_a^2, but we are only interested in its encouragingly small ratio to $\dot{\Sigma}_{0,0}$ and $\Sigma_{0,0}$, for which purpose σ_a^2 is also irrelevant.)

How efficiently can this approach be used? It might appear at first that we would need $\Theta(l^3)$ computation per interval at each processing element, in order to compute the matrix product $\widehat{\mathbf{\Sigma}}\mathbf{\Sigma}^{-1}$. However, on closer examination the actual figure is $\Theta(l)$, which is much more reasonable. The only parameter affecting $\widehat{\mathbf{\Sigma}}\mathbf{\Sigma}^{-1}$ which isn't known until run-time is k. What we can do is precompute $\widehat{\mathbf{\Sigma}}\mathbf{\Sigma}^{-1}$ for each possible value of k. (In principle, k could be as large as n, but the probability of this is vanishingly small, so for large n a much lower upper bound could be used in practice; call this k_{max}.) The last row of each of these matrices can then be stored in a $k_{max} \times l$ table for use at runtime. When $\hat{z}_0$ is needed at runtime, the dot-product of the k^{th} row of this table and $\mathbf{x} - \dot{\mathbf{z}}$ is calculated and added to $\dot{z}_0$. This takes only $\Theta(l)$ time, as promised.

We would like, after all this effort, to have arrived at a substantially better estimator of z_0 than either $\dot{z}_0$ or x_0 is. (Recall that $\dot{z}_0$ is the estimate derived from the time-series model using only observations old enough to be assumed accurate, while x_0 is the estimate derived from the averaging algorithm alone, without use of the time-series model.) To reassure ourselves on this matter, we calculated representative values for $\Sigma_{0,0}/\widehat{\Sigma}_{0,0}$ and $\dot{\Sigma}_{0,0}/\widehat{\Sigma}_{0,0}$. Taking $m = 3$ (the optimum), $\theta = -0.4$ (from our simulation experiments with ELINT), $n = 64$ (ditto), $l = 6$ (ditto), and $k = 1 + m/2$ (for the most realistic comparison), we get $\Sigma_{0,0}/\widehat{\Sigma}_{0,0} \approx 7.81$ and $\dot{\Sigma}_{0,0}/\widehat{\Sigma}_{0,0} \approx 3.30$. Thus, by using the Bayesian inference procedure to incorporate the recent data we have better than a three-fold improvement in the quality of our time-series forecast. Conversely, by using it to incorporate the time-series model's analysis of historical data, we hope to achieve more than seven times the precision we could have achieved had we relied only on current data. For larger n, the recent data isn't as valuable, but still worth incorporating: for $n = 1024$, increasing l suitably to 9, we get $\Sigma_{0,0}/\widehat{\Sigma}_{0,0} \approx 56.0$ and $\dot{\Sigma}_{0,0}/\widehat{\Sigma}_{0,0} \approx 2.28$. As might be expected, the payoff from using historical information is greater, since the four sites known about are less representative of the 1024 than they would be of 64. However, by optimally incorporating the recent data in with the historical, we still get a further two-and-a-quarter times improvement in precision.

One final technical difficulty that shows up in our actual implementation is that rather than having some constant number k of averages of each age, we may have a varying number. This arises because any processing element whose housekeeping processor's load exceeds an emergency threshold stops performing inessential load-balancing activities. This has two impacts: firstly, we may on rare occasion have slightly fewer averages for one or more past intervals than for the remainder; secondly, we may somewhat more commonly have extra loads for the current interval.

Taking the former complication into account would require a rather expensive generalization of our inference procedure for an exceptional condition. Therefore, the implementation correctly averages the reduced number of averages, but then weights that average-of-averages as if there had been the full complement. The latter complication—additional current observations—is not only more common, but also easier to incorporate into the analysis. All that is necessary is to follow the above-described multivariate Bayesian inference by an analogous univariate one.

4.5 Partner seeking

Our algorithm integrates the finding of a partner for object migration with the above method of estimating the average load. When a processing site receives one of the multicasted load messages, it considers it as a possible offer of work, as well as a source of refining load information.

The receiving site estimates the current average load, using the new information from the message as well as all prior information. It then compares that estimate of the average with both its own current load and the load of the sending site, which is included in the message. If the receiving site is underloaded by more than a threshold amount and the sending site is overloaded by more than a threshold, the receiving site sends back a request for work. In the meantime it reserves the fraction of its underload that it hopes to have satisfied, in order that it doesn't request work from multiple sources and have too much arrive.

The multicast breadth of three calculated above as optimal for average determination is rather narrow for finding partners. Therefore, if the site receiving a load message finds that the sender's overload not only is sufficiently large but also exceeds the receiver's underload (if any), it multicasts a request for "reinforcements" to a couple of extra randomly-chosen sites. Of course, it only sends this request for reinforcements if the residual overload exceeds the appropriate threshold and the estimated average load is high enough to make a sufficient underload possible.

4.6 What to migrate

The experimentation described later in this thesis used a very simple greedy heuristic to determine which objects to migrate to shift a desired amount of load. The heuristic attempts to minimize overhead, rather than trying to respect priorities. (Chang has considered the latter alternative in [15]. He compared algorithms that attempted to maintain a global priority ordering with one that scheduled locally by priority but transferred jobs in a priority-blind fashion, and found that "Surprisingly, [the latter] performs well in supporting priority jobs ...." [15, p. 38])

Those objects at the overloaded site which have messages awaiting reception, with the exception of the one currently executing, are ordered by the number of waiting messages. The currently executing object is excluded from consideration because it is quite literally executing, since a separate housekeeping processor (the operator) is used for the load balancing. The candidate objects are considered one at a time, from most messages to least. If an object can be included in those to be sent without overshooting the target load (total number of messages), then it is selected for inclusion. This continues until either all eligible objects have been considered or the desired total load is exactly achieved.

This tends to result in a relatively small number of objects being migrated, which reduces the overhead. Of course, not all objects will be equally large. However, just as the processing time for a message is not *a priori* known, so too the size of an object is not easily available. This somewhat surprising fact results because the instance variables of an object may have arbitrary graph-structured, dynamically-allocated values. (The Lamina programming language is not purely object oriented; it also allows dynamically allocated structures within each object's state.) In any case, not all of the object migration overhead is tied to the size of the object's state; redirecting inter-object communications is also costly, and relates to the number of active communication partners the object has.

Even ignoring these issues and also the priorities of the objects, it would be possible to use fancier selection criteria to try to more nearly equal the desired amount of load. However, this would itself add overhead at an already overloaded site. Therefore, no attempt has been made to go beyond the simple but effective greedy object selection heuristic.

4.7 Summary of modeling and algorithmic contributions

This chapter has laid out the components of our load balancing algorithm. More importantly it has shown how explicit models of the randomized information dissemination process and of the evolution of the system's load can be used in the design of the central load-estimation component of our algorithm.

By using these models, all available information on system-wide loading can be properly combined to yield an accurate estimate of the current load. The model of the averaging process quantifies the extent to which older information is more representative of the system as a whole, while the time-series model of the load provides the statistical connection between loads of various ages, including in particular the current load. A multivariate Bayesian inference technique is used to combine these sources of information.

Beyond this central contribution of the thesis, this chapter has also shown how the efficiency of the randomized distributed averaging algorithm can be optimized by using the asymptotic analysis of that algorithm to balance its progress per interval against the cost of the intervals, another original result.

Finally, this chapter has briefly addressed the issues of load metric (queue lengths), partner seeking (integrated with the load information multicasts), and object selection (greedy).

Chapter 5

Implementation Issues

The previous chapter described in general outline our proposed algorithm for load estimation and global load balancing. However, it left open many matters of implementation detail:

- How fine a time division are the load-balancing intervals?
- What assumptions are made about clock synchronization?
- How large an overload or underload warrants corrective action?
- Should messages not yet consumable due to sequence-number constraints count towards load?
- What if an operator is itself too overloaded to keep up with the load-balancing overhead?
- What if load-balancing messages are delayed a long time in transit?
- How are objects encoded into a linear form for migration?
- How is inter-object communications redirected when a recipient migrates?

This chapter describes our experimental implementation of the algorithm in order to provide one possible set of answers to these questions. This chapter also describes the cost model for this (simulated) implementation, i.e., how long each operation takes, which provides background for the performance results in the next chapter. These two chapters constitute a case study of how the general load-balancing strategy of this thesis can be applied. In order to put the quantitative aspects of this case study in perspective, including in particular the operation times given in this chapter, it is necessary to specify the performance of the simulated CARE multicomputer. The evaluators are assumed to be 32-bit processors; their performance was crudely characterized as "10 MIPS" in [19]. The operators' performance is such that they can encode arbitrary pointer-based structures for transmission in 1.6μs per 32-bit memory word. Each link in the network is 16 bits wide, and a 16-bit half-word can be transmitted one hop in 200ns.

5.1 Timer-driven activity

The analysis of section 4.2 allowed us to optimize the performance of the load-averaging process subject to a fixed limit on its resource consumption. However, it still remains to decide just how much of the system's resources should be allocated to this process; this is controlled by setting the algorithm's time interval.

Experimentation with ELINT prior to the implementation of our load balancing algorithm had yielded a rough understanding of the rate at which the system loading changed (hence how long an interval could be tolerated without sampling the load too coarsely) and of the amount of excess operator and network capacity (hence how much extra load the averaging process could sustainably impose). This prior empirical data was only a rough starting point, because the introduction of the load-balancing algorithm could change the rate at which the system's load changed, for example.

Some back of the envelope calculations were done concerning the relationship between the interval and the operator and network utilization imposed; for example, we show below that an interval of 2.5ms causes the operators to spend about 8% of their time doing load estimation. Another consideration was that it is desirable that the load sampling interval evenly divide the period of the input data, so that each load interval can be matched to corresponding ones in previous input periods.

Based on these considerations, we chose 2.5ms as the basic load-balancing interval in our implementation. That is, all load estimates are averages for a 2.5ms time interval, and the multicasting is done at 2.5ms intervals. Each operator uses a local interval timer to trigger itself every 2.5ms to do a multicast. These timers are intentionally not synchronized (in fact, are offset by an initial random delay), so that the multicast activity is spread over the entire interval. However, it is assumed that the operators have clocks that are synchronized to better than 2.5ms accuracy, so that they can tag load estimates with the interval to which they apply.

When an operator's timer goes off, it uses the total number of queued tasks (messages at objects) as the local load metric. The cost model assumes that this is available without actually traversing the various queues, since it is possible to simply increment and decrement a counter as tasks are enqueued and dequeued.

One interesting complication arises from the fact that LAMINA objects are allowed to impose a sequence-number control over their incoming messages [21]. That is, an object may require that messages be tagged by their sender with consecutive integers; if a message arrives out of order, processing is delayed until the earlier messages have arrived and been processed. When counting queued messages, should messages in this waiting condition be counted? The actual implementation does count them, because they normally will be ready for processing very soon, so their inclusion produces a more realistic estimate of the available load. However, this had to be weighed against certain dangers in making this choice. It is possible that the sender is intentionally not generating messages in sequence number order, for example to implicitly do sorting. Also, there is the risk that an object with only waiting messages queued could in theory forever "outrun" the rest of its messages if it were allowed to migrate, thus causing a portion of the computation to be starved for data. (Had we made the other choice and not included out-of-sequence messages in the load, our algorithm would provably never cause this kind of starvation, since an object is only migrated if it has messages queued.) Both of these dangers seem rather remote.

In our simulation model, the processing done when the timer goes off normally takes $46\mu s$ if any information for the current interval has already been received; otherwise an additional $3.5\mu s$ is spent initializing the interval. This latter is a once-per-interval cost, like the interval timer and multicasting; in the case where it isn't incurred when the interval timer goes off, it is because it has already been incurred when the first message of the interval was received. Therefore it makes sense to lump these times together as about $50\mu s$ per interval at each site. This cost is actually computed in the simulation model using a linear function of such variables as the multicast breadth and the number of load balancing intervals for which the estimates are improved. Here as elsewhere in this chapter we are simplifying to a single time by assuming the parameters as used in the experiments described in the next chapter. Where there is variation, the numbers will be based on the normal case, with remarks about major sources of variation.

Sometimes the operator is so busy that any unnecessary work must be avoided; this is what the implementation terms *panic mode*. In the experiments described in this thesis, the threshold for panic mode is set to be an operator load (queue length) of three. The timer-interrupt processing described above is only slightly different in panic mode, because the heavy operator load is likely caused by heavy evaluator load, and thus it makes sense to still advertise the local load in hopes of relief. Therefore, all that is different is that the load history is not updated or included in the transmission and the cost of preparing for a new interval isn't incurred. Instead, the current local load is all that is multicast to the randomly selected sites. This amounts to a savings of over $15\mu s$, largely the result of the transmitted packet being smaller, since in needs only to include the one current load rather than also six past loads.

5.2 Summary of message types

Other than this timer-interrupt service that triggers the multicasting, all other activity is data-driven, triggered by the arrival at an operator of an appropriately tagged message. Four different message types are used:

load-information This is the message multicasted in response to the interval timer. It contains an indication of the originating site, time sent, that site's load at that time, and unless the originator was in panic mode a vector of historical load estimates. The recipient uses it both as a source of load information and as a potential offer of work.

work-request The work-request message is sent in reply to a load-information message if there appears to be sufficient overload and underload to warrant migrating some objects. In addition, a work-request message can be triggered by a relief-request message from a third party.

relief-request The relief-request message is also sent in response to a load-information message when the load-information message was sent by an apparently overloaded site, but rather than being targeted to that overloaded site it is multicast to two randomly selected other sites, asking if they can take some work from the overloaded site. This is only sent if there is there appears to be a sufficient excess of overload beyond the local underload and a sufficiently high average load. The message includes the estimated remaining overload and the estimated average load.

work-migration When the overloaded site gets a work-request message, it responds with a work-migration message containing the objects chosen to be migrated. Even if the overloaded (or previously overloaded) site chooses not to send any objects (for example, because it is no longer overloaded) it sends an empty work-migration message so as to cancel the *underload reservation* at the recipient site. This frees the underloaded site to seek work elsewhere without fear of receiving too much.

A typical scenario showing all of these message types in use is illustrated in figure 5.1; the following sections go into more depth on how each message type is processed.

5.3 Load-information processing

If the operator receiving a load-information message is in panic mode, it simply discards the message without any processing at all beyond inspecting the tag. Otherwise,

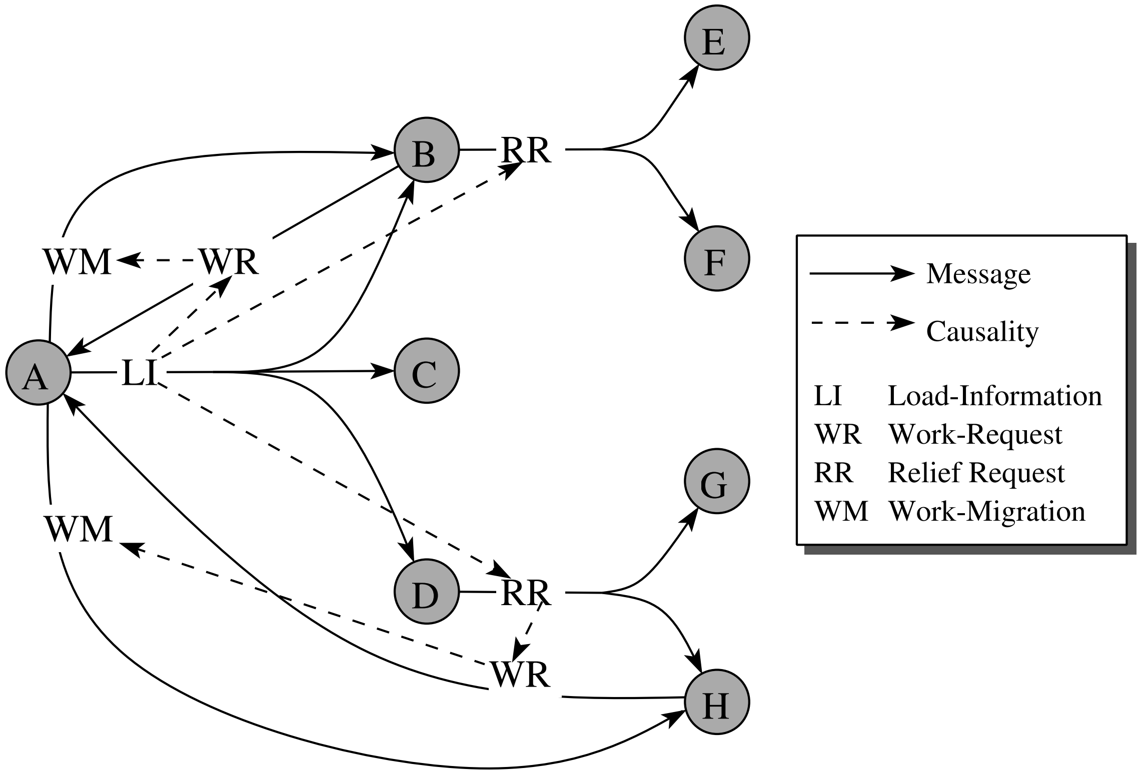

Figure 5.1: Load balancing messages. In this scenario, site A has been triggered by its interval timer. It multicasts a load-information message, choosing sites B, C, and D as the recipients. Site B decides that A is overloaded and B underloaded, so B sends A a work-request message. However, B is not nearly so underloaded as A is overloaded, so B also multicasts a relief-request to two sites, chosen to be E and F. Meanwhile, C generates no messages at all in response to the load-information, perhaps because it is in panic mode, or because with its estimate of the average load A doesn't appear overloaded. D recognizes A to be overloaded, but is not itself underloaded. Therefore, it sends no work-request but rather only a relief-request to G and H. Of the four sites receiving relief-request messages, only H is underloaded, so it sends a work-request message to A. A responds to each of the two work-request messages it gets with a work-migration message, one to B and one to H.

the load information is merged into the recipient's own load history.

Next, if the packet has been in transit for more than 2.5ms, it is discarded; this allows a "logjam" of messages to be more quickly cleared, so as not to cause further congestion. (Because the CARE routers are designed to be simple and fast, they can not discard stale messages in transit; thus this task falls on the receiving operator.)

Otherwise, the message can be interpreted as an offer of work as well as a source of load information. The recipient estimates the system-wide average load, and checks whether the sender was overloaded by at least two tasks. (This threshold clearly needs to be at least one task to ensure stability, since that is the granularity for work transfer. We chose the more conservative threshold of two to ensure that even in the face of small variations in the various sites' estimates of the system-wide average load, there is little risk of load being shuffled back and forth repeatedly.)

If the sender was sufficiently overloaded, the recipient checks whether its own load is below the estimated average by at least two tasks. In doing this, it doesn't include any underload that is expected to be filled by other sites to which work-request messages are outstanding. However, these underload reservations expire after 2.5ms if there is no response to the work request. Furthermore, the recipient need not be underloaded by two tasks if it is completely idle, so long as the estimated average load is at least one—thus load migration can continue even if the estimated average load is less than two.

Assuming all these tests are passed, the recipient operator now sends a work-request message to the site from which the load-information was received. The amount of work requested is the smaller of the estimated remote overload and the estimated local underload (as adjusted for outstanding work-requests). This amount isn't actually included in the work-request message, but it is calculated because this amount of underload is recorded as reserved, in case a similar load-information message is received before the work-migration.

The work-request includes the requesting site and its estimated adjusted underload, a reservation id number for the underload reservation, the estimated average load (so the overloaded site can re-assess its overload without re-estimating the average load) and the current time (so old work-requests can be ignored). The specific amount of work to transfer isn't included, because the other site may in the meantime have successfully transferred work elsewhere.

If the estimated remote overload exceeds the adjusted estimated local underload by at least two tasks and the estimated average load is at least one, then a relief-request message is also multicast to two randomly-chosen sites. This message contains the estimated remaining overload, the estimated average load, the time, and the overloaded site to which any work-request should be directed.

The total time for this processing can range from about 50μs if the remote site doesn't appear overloaded up to 85μs if the work-request and relief-request both prove necessary. Thus, even if no work-requests or relief-requests ever get generated (and hence no work-migrations), the combined cost of the timer processing and load-information processing amounts to roughly 8% of the total operator time, because $(50\mu\text{s} + 3 \times 50\mu\text{s})/2.5\text{ms} = .08$.

5.4 Relief-request processing

The processing of relief-request messages is simpler. As with load-information, the message is immediately discarded if the operator is in panic mode or if the message is older than 2.5ms. The estimated average load contained in the message is used to calculate the estimated local underload, adjusting for outstanding work requests (i.e., underload reservations not older than 2.5ms). If this underload is at least two, or alternatively the local evaluator is completely idle, then a work-request is sent and an underload reservation recorded. The amount of work reserved is calculated as the smaller of the estimated local underload and the estimated remaining remote overload (i.e., the overload minus any work requested by site originating the relief-request). The work-request goes directly to the overloaded site, rather than to the site from which the relief-request was received. Because no average estimation is required, this process is relatively cheap; if a work-request is generated, it takes about 30μs; if not, less than 20μs.

5.5 Work-request processing

The work-request message isn't ignored in panic mode, because the panicking site may be able to get out of panic mode precisely by offloading objects. (Recall that after redirection has occurred, the communications as well as computation associated with the migrated objects will be gone.) However, work-request messages that have been in transit for more that 2.5ms are still ignored, because it could lead to dangerous oscillations if work were migrated based on stale information.

The work-request message contains the average load as estimated by the requesting site. This is important not only because the overloaded site shouldn't have to waste precious resources estimating the average, but also because it may not have the information on the basis of which to estimate the average if it has been in panic mode.

The site receiving the work-request message calculates its overload using the estimated average load from the message and compares it with the estimated underload of the other site, also taken from the message. Whichever of these two is smaller serves as the target number of tasks to transfer. The greedy policy described in section 4.6 is used to select specific objects to migrate so as to transfer at most that many tasks. Those objects are then sent together with the reservation id number from the work-request in a work-migration message targeted to the site from which the work-request came.

Except for the case where the message is ignored, the minimum processing time is when the site is no longer overloaded and hence doesn't need to select or transmit objects. In this case, the processing takes 35μs. The extra cost above this varies substantially, depending on the number of objects, how many of them are selected for transmission, and how much state those objects contain. The final encoding of object states for transmission is accounted for at 1.6μs per 32-bit word. (If that seems high, recall that the object states involve arbitrary pointer-based structures, and that this is in the context of a 10MIPS processor.) The experimental results on migration delays presented in section 6.8 includes these times as well as actual communication time.

The process of actually encoding the objects for migration is rather difficult because the LAMINA language is not purely object oriented. Thus the instance variables may contain not only references to other objects and scalar quantities but also references to general records and arrays which may form an arbitrary directed graph structure. Therefore, what is needed is a graph-traversal algorithm that can tolerate cycles and can efficiently produce an encoding that reflects any structure sharing. Luckily the LAMINA/CARE system already provides operator facilities for passing general directed graph structures by copying [54], so only a few superficial modifications were necessary to accommodate migrating entire process states.

5.6 Work-migration processing

Not surprisingly, work-migration messages are the one type of load-balancing message that is never ignored. The receiving operator removes the indicated underload reservation if it hasn't already expired, decodes each arriving object (turning relative addresses into absolute pointers, for example), initiates the process of redirecting communications to each arriving object, and installs each object in the evaluator's queue of runnable processes. The bare minimum time for processing this type of message is $11\mu s$, if there are no objects in the message and the reservation id is found immediately at the head of the list of reservations. Decoding objects takes $1.6\mu s$ per 32-bit word, creating a process takes $15\mu s$, and redirecting communications takes a minimum of $16\mu s$ for the initial message sent (more work ensues later at a variety of operators, including the originating one). The following section explains how the redirection of communications is handled, which is the most complex aspect of the process.

5.7 Communication redirection

All communications between objects in LAMINA is mediated by explicit *streams* managed by the operators. There are provisions for forwarding one stream to another, and for updating upstream streams to point directly at new streams added further downstream [21]. These facilities needed only slight extension to facilitate object migration. Objects now have local outgoing streams forwarded to the receiving objects' incoming streams. When an operator receives a migrated object, it creates a new incoming stream for the object and requests that the sending operator forward the old incoming stream to the new one. This causes any waiting messages to be immediately dispatched to the new location and also leaves a forwarding address for any messages that arrive thereafter. (This is for the primary incoming stream of the object; the object may have additional incoming streams, which are similarly forwarded later if the object waits on them.) The first message sent to the migrated object by any other object will be forwarded from the outgoing stream of the sender to the old incoming stream at the former location and from there to the incoming stream at the new location. When this happens, a message is asynchronously sent to the originating outgoing stream changing it to forward directly to the new incoming stream; if the original message had to take a multi-hop forwarding path, the intermediate streams are also similarly updated. In early experiments we immediately asynchronously updated all of the outgoing streams but this proved to be quite expensive, as there are often many potential senders, few of which will actually send in the near term. Moreover, maintaining and migrating the back-pointers to the senders was itself very costly. On the other hand, the current policy of only updating addresses when they are used increases the risk of longer forwarding chains, particularly if the migration rate is high. Preliminary experimentation with ELINT demonstrated a net gain from switching to the current policy, but we don't present any quantitative results to substantiate this, since communications redirection was not the primary focus of this dissertation. Others have studied this issue more carefully; in particular, Fowler [28] in his examination forwarding address policies provides asymptotic amortized analysis for path compression techniques such as we chose to use. His analysis confirms that our choice of path compression is reasonable, by showing that the total number of messages required for m migrations, a message deliveries (with $a > m$), and N processors is asymptotically bounded by $a + 3a \log_{1+a/m} N$.

5.8 Summary of implementation issues

This chapter, by providing a case-study description of our experimental implementation of the load-balancing algorithm, has served two purposes: it has provided concrete examples of decisions made in the course of putting the general approach into practice, and it has described the experimental context in which the results of the next chapter were obtained.

The implementation uses an interval timer at each site, all with a common period but with randomized phases, to initiate load information dissemination. The load information messages can also serve as offers of work, and as such can cause work request and thus work migration messages to be sent and processed. A “relief request” message type is also employed when it is appropriate to search for additional underloaded sites to come to the aid of a particularly overloaded site.

We’ve shown the operator processing costs associated with all these actions in our implementation; given these costs, our choice of a 2.5ms interval corresponds to a decision to devote about 8% of the operators’ time to load estimation. We’ve also described the “panic mode” and timeout mechanisms used to respond to operator overload or network congestion in a stabilizing negative-feedback manner. Finally, we’ve sketched how the pre-existing mechanisms of the CARE implementation of LAMINA are used to provide relocation of object states and redirection of communication.

Chapter 6

Performance Results

There are several inherently empirical questions raised by the load balancing methodology proposed in this dissertation:

- Can the loading in real systems be adequately modeled?
- Are the estimates of the system-wide average load generated through time-series forecasting and Bayesian inference actually superior to those achieved without one or both of those techniques?
- Do those improved estimates allow the load to be balanced with fewer object migrations and with fewer objects being repeatedly migrated?
- What is the net change in overhead costs associated with more complicated load estimation but less frequent object migration?
- What impact does the frequency of object migrations have on application performance?
- How long does it take to migrate an object? To negotiate a migration? Given these, should the time-series model be used to forecast future load, rather than estimate current load?
- How does the initial object placement strategy interact with the object migration strategy?

- How do applications fare that have static grids of objects communicating with their neighbors?
- What impact do such factors as the relative cost of migration and the overall system loading have on the conclusions?
- Finally, putting all of the above together, under what circumstances is net application performance improved by using the methodology of this dissertation?

The first question, whether actual systems' loads can be successfully modeled, was answered affirmatively in section 4.3, at least for ELINT and AIRTRAC-DA. This chapter answers the remaining questions using experimental evidence from simulation. This experimentation complements the modeling and algorithmic contributions presented earlier, and is itself a major contribution of the thesis. The experiments compare our proposed load-balancing method with several alternatives, described in the first section.

The net impact on application performance is the ultimate measure of success; so as not to keep the reader in suspense any longer, immediately after listing the alternative methods we will examine data concerning the application-level performance of ELINT. Having done so, we can then look more closely at the ELINT runs for such particulars as the quality of load estimates or the performance impact of migration frequency, with an eye to explaining the application-level performance effects we have observed. Finally, we will introduce a synthetic application of the highly regular nearest-neighbor sort to see how it fares under various combinations of object placement and object migration strategies. All these results are from experiments with a 64-site configuration of CARE.

6.1 Alternative load-balancing policies used for comparison

The ELINT experiments compare four different migration policies, all using random initial placement of the objects. In addition to these four, the ELINT comparisons also include a more sophisticated deterministic placement strategy without any migrations. The synthetic-load experiments described in section 6.10 use a somewhat different mix of placement and migration strategies, which will be described in that section.

The four migration policies used in combination with random placement are as follows:

- The global load-balancing scheme proposed in this thesis uses time-series analysis and Bayesian inference to estimate the system-wide average load. Load is transferred between two sites if each differs from the estimated average by more than a threshold amount (in opposite directions); as a special case, a completely idle site is always eligible to receive work, even if the estimated average is less than the threshold amount. The amount of load transferred is the smaller of the differences from the estimated average. This will be called the "Bayesian" scheme in the comparisons.

- A simplified version of the same global load-balancing policy uses only load data from the current time interval to estimate the average load, rather than using the techniques of this thesis to incorporate any historical data. This will be referred to as the "recent information" scheme.

- A yet simpler load balancing scheme uses only the current loads of the prospective donor and recipient sites, rather than any estimate at all of system-wide loading. Objects are migrated if the difference between the two sites' loads exceeds a threshold equal to the sum of the underload and overload thresholds in the other migration strategies. The amount of load transferred is half the difference in load of the two sites. This is a "local" load balancing scheme in the broad sense that the migrations serve to locally balance the loads of the two sites in question, rather than being calculated to bring one or both of them to the system-wide average, thus contributing to global load balance. However, the same non-local randomized communication pattern as in the other schemes is still used, so the two sites locally balancing their loads need not be nearby to one another. This scheme will be referred to as the "averageless" scheme.

- Finally, a version that does no load estimation or object migration at all is included, which will be referred to as the "random static" scheme. The objects are simply left where initially randomly placed.

In addition to these four options for deciding when (if at all) to migrate the randomly placed objects, the ELINT experiments include another load balancing policy which deterministically places the objects and also does no migrations. This is a policy that was designed by Saraiya in the course of his earlier experiments with ELINT [54]. In this policy, the sites are partitioned into separate regions for the various classes of objects. Saraiya chose the relative size and positions of the regions so as to produce both good load balance and low-congestion communications patterns. In the following comparisons, we will call this the "class-based static" scheme.

6.2 Application latencies

For a soft-real-time surveillance application such as ELINT, the appropriate performance measure is not throughput but rather *latency*, i.e. the delay between the arrival of sensory data and the output of the corresponding report. Further, it doesn't suffice to measure the average latency, since what is desired is consistently low latencies for all output reports. These experiments used the two most common kinds of output reports generated by ELINT: the fix (i.e. position) and heading of individual "emitters," which are sources of radio frequency emissions, such as missile guidance radars and identify-friend-or-foe devices.

All experiments were done with a fixed input file of simulated observation inputs. The inputs were scheduled to arrive in periodic bursts. The interval between bursts—the "data period"—was held constant over each simulation run. Two different data periods were used, each for an entire group of runs, in order to show the impact of the overall load level. Figure 6.1 shows the number of input observations each data period in the input file used for these experiments.

Because the latencies depend not only on the object migration method, but also on the initial object placement, we took care to use a variety of initial placements and to use that same assortment of placements for each of the migration methods.

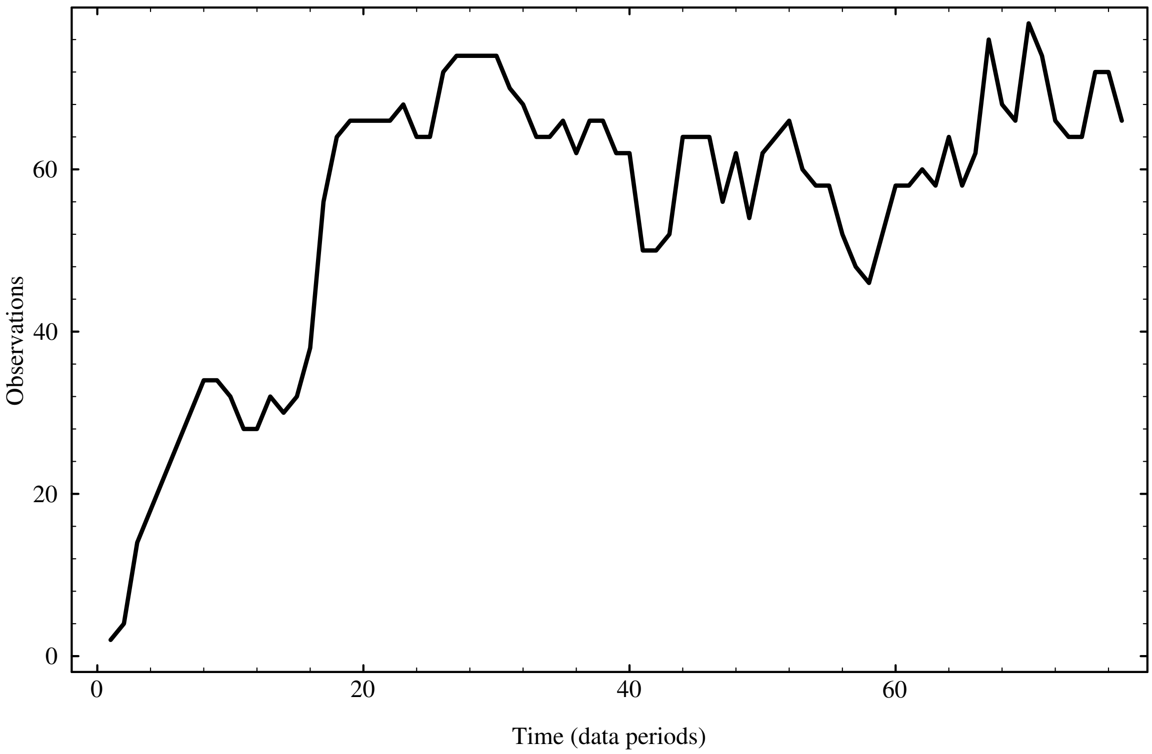

Figure 6.1: Input data. This graph shows the number of input observations each data period in the data file used for the ELINT experiments.

As noted above, except for the one policy which combines deterministic placement with no migration, the alternatives being compared all used random initial placement. Therefore, the desired variety of initial placements was achieved by doing seven runs with each load balancing strategy, using different seeding of the pseudo-random number generator—the same seven seedings with each competitor. As a side effect, what was varied in this controlled manner was not only the initial object placement, but also the particular pseudo-random sequence used for randomized communication of load information. This helps insure that the comparisons are not unduly influenced by particularly fortuitous pseudo-random sequences. For the one deterministic policy (Saraiya's class-based static placement), only a single run was necessary.

We measured the fraction of emitter fix and heading latencies which exceeded each of a range of thresholds and plotted this miss rate against the threshold, measured in multiples of the data period. This is essentially the same as a cumulative distribution function for the latencies, and allows easy visual comparison of our load balancing method with alternatives. These miss rates for the case of a 15ms data period are plotted in figure 6.2. The miss rates shown are averages over the seven runs for each method (except the class-based one).

It's quite evident that dynamic migration substantially improves the system performance. The three dynamic schemes cluster quite closely together; if one looks very closely, the apparent order amongst them is that the averageless scheme is worst, the recent information scheme best, and our own Bayesian scheme in between. The difference among them is so small, however, as to bring this ordering into question. We show below that it is of marginal statistical significance; however, even if significant in the statistical sense, it is of no practical significance.

It is interesting to notice that the class-based static scheme actually performs substantially worse than random static placement of objects. Yet Saraiya carefully hand optimized the placement of the various object classes so as to achieve both good load balance and low-congestion communications patterns. How then is it bested by random allocation? The key is that Saraiya did all his work in the context of another single input file, and tuned his performance around the characteristics of that scenario. It turns out that this placement was brittle—that what worked very

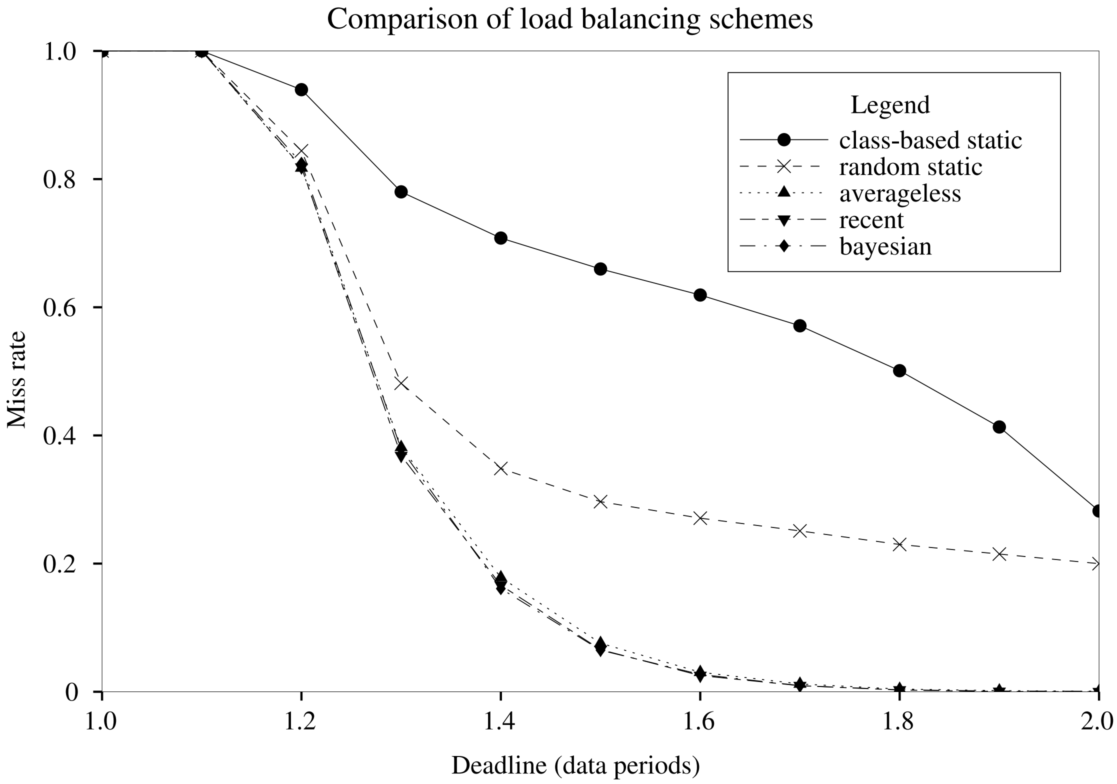

Figure 6.2: Comparison of load balancing schemes. This graph shows the fraction of emitter fix and heading reports missing each of a range of deadlines, shown on the x-axis in units of the data period, here 15ms.

well for one scenario works quite poorly under a different scenario. This helps explain our focus on simple adaptive, dynamic, randomized algorithms—although they are unlikely to be optimal in any particular situation, they are equally unlikely to be pessimal in another (unanticipated) situation.

6.2.1 Increased migration cost

Although the three dynamic load-balancing schemes exhibit comparable performance under these conditions, there may be important differences among them. In fact, the more detailed evidence presented below shows that this three-way tie is actually a coincidence: the dynamic schemes succeed to varying degrees in balancing the load, but also exact greatly varying migration costs and load estimation costs, and these effects—particularly the latter two—happen to roughly cancel out.

Therefore, it is interesting to see what would happen if the relative cost of migration was changed, for example by inflating the size of the process states by a factor of five. This serves to illustrate the sensitivity of the analysis to architectural variations affecting relative speeds of processing and migration, as well as to variations among applications in the ratio of object size and method complexity. As a guideline for comparing our two experimental situations with other systems, it is helpful to know that data later in this chapter shows that the uninflated process sizes result in migration times approximately equal to the method execution times.

If we do this experiment, we get the results shown in figure 6.3. This graph was constructed in the same manner, again using seven runs from each scheme except the class-based one. (The static schemes are of course unaffected by the size of process states, since they do not need to migrate them.) Here the differences between the various schemes' performances are more substantial, and the scheme proposed in this thesis shows the best performance.

Another point worth noting in this graph is that it is ambiguous whether the migrations done by the averageless scheme are hurting performance as much as they are helping it. For short deadlines, the miss rate is worse than with the migrationless random static scheme, though for longer deadlines the miss rate is lowered by the migrations. Because the averageless scheme has the loosest standards for when to

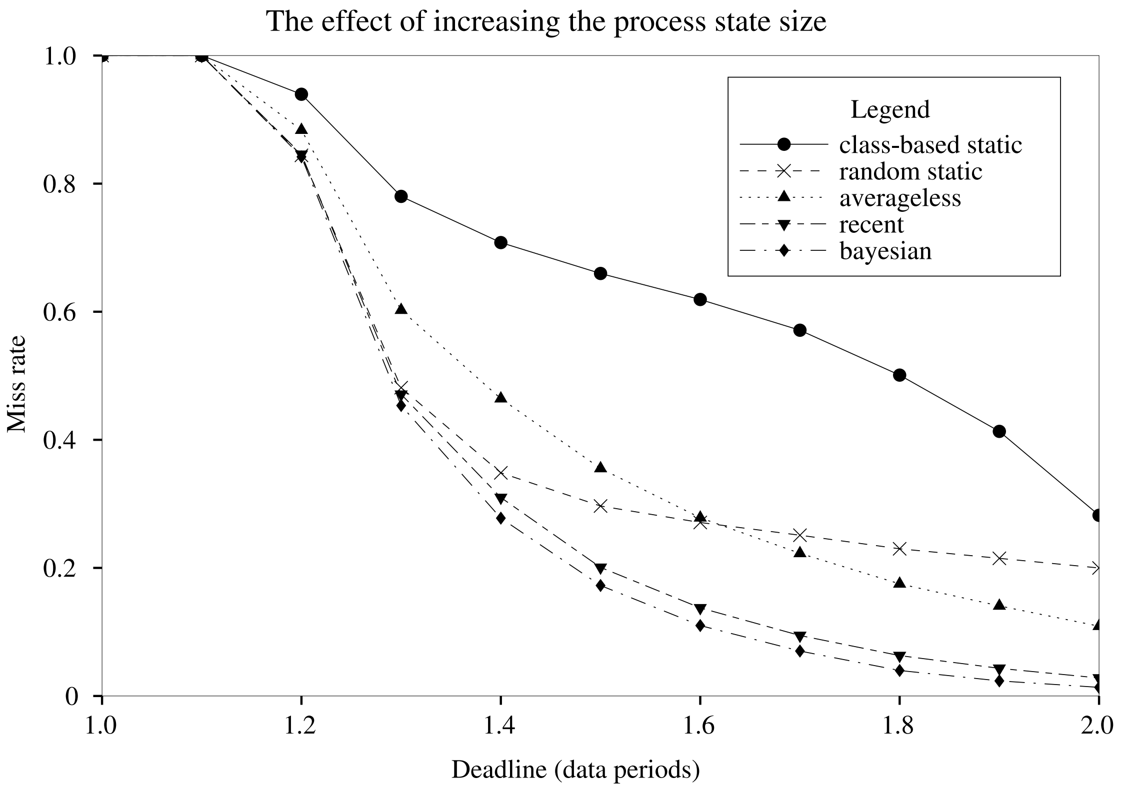

Figure 6.3: The effect of increasing the process state size. This graph again shows the fraction of emitter fix and heading reports missing each of a range of deadlines, shown on the horizontal axis in multiples of the 15ms data period. This time the process state size is five times larger.

migrate objects, it winds up doing the most migrations, as we'll see below; this results in the scarcity of short latencies. However, it still beats the random static scheme at high thresholds, because that latter scheme suffers a flattening out of its miss rate curve due to bottlenecks at overloaded sites. (Those reports which require processing on the overloaded sites are subject to extremely long latencies.)

6.2.2 Statistical significance

Even though the miss rate curves are now clearly distinguishable, it is worth testing whether those differences are statistically significant. Recall that the miss rate curves reflect averages; because the actual latencies vary substantially, the lines should be thought of as rather broad. Are they so broad as to render our scheme's apparent superiority insignificant?

Suppose the recent information scheme actually was identical in performance to our scheme; this is our *null hypothesis*, which we will test in hopes of refuting it. In that case, which of these two migration schemes is used shouldn't make any difference in the latencies of the output reports. However, the latencies may none the less vary considerably depending on the initial object placements, independent of the migration scheme. Further, some input observations may be responded to more slowly than others, independent of the migration scheme. Therefore, a *paired* test is called for, which matches each output latency observed using one migration scheme with the corresponding output latency from the other scheme. The corresponding latency is the one for the same input observation and using the same initial object placement.

If we pair up the observed output latencies in this way, in some pairs one latency will be larger, while in others it will be the other way around. If there is a surprising degree of asymmetry, one unlikely to be due to chance, then the null hypothesis can be rejected in favor of the alternative that the two migration schemes differ in performance.

One way to examine this visually is using a histogram of the pair differences. It is difficult to see the asymmetry in this histogram, presented normally. However, if we eliminate the central bar (reports where the two methods had nearly identical latencies—somewhat over a third of the total reports) and fold the remainder of the

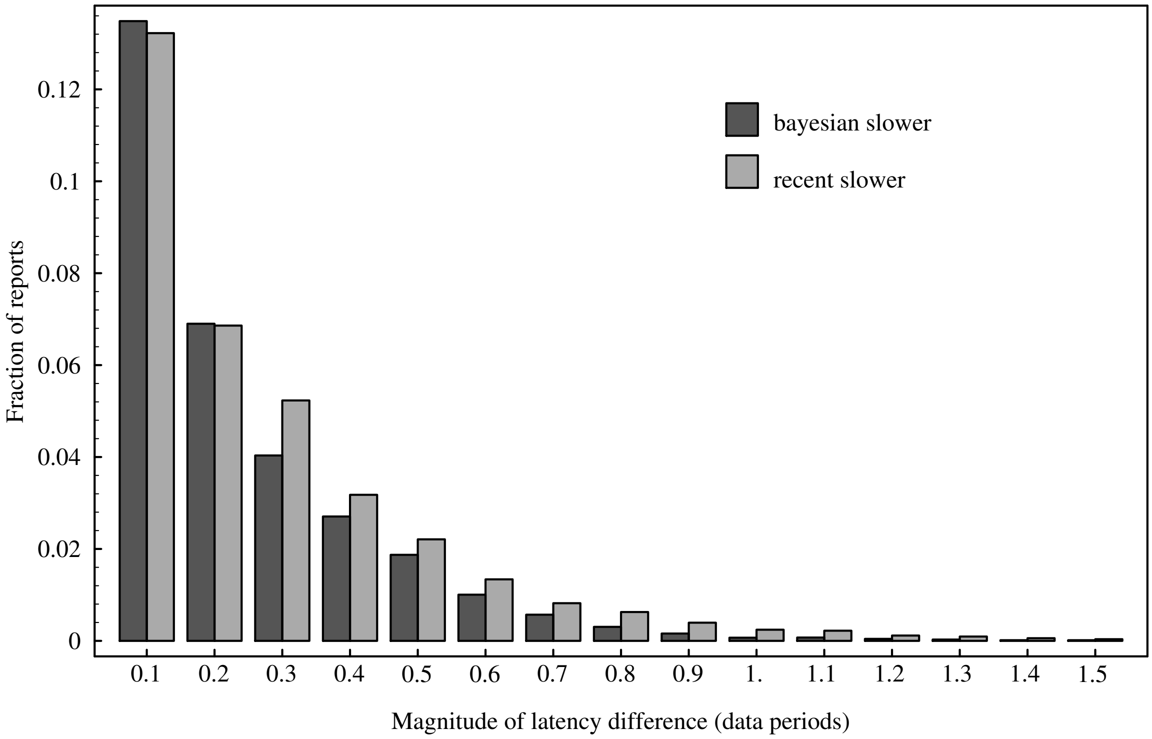

Figure 6.4: Folded latency difference histogram. This shows the distribution of differences between latencies using the proposed Bayesian scheme and the corresponding latencies using the recent-information scheme with the same placement and for the same observation. This figure excludes the reports where the latencies differed by less than .05 data periods, and places the positive and negative latency differences of each magnitude side by side. This figure is from the experiments with 15ms data period and the five times larger migration cost.

histogram in half, so that corresponding bars from the positive and negative sides are adjacent, as it figure 6.4, then the asymmetry is quite clear. There is a substantial excess of cases where the recent-information scheme was slower. Although for some individual reports the Bayesian scheme proposed in this thesis was slower, it would be remarkable if this happened so infrequently by chance alone.

The probability that this asymmetrical distribution would in fact result by chance can be quantified, producing a significance level at which the null hypothesis is rejected. More precisely, we are computing the probability of a distribution this asymmetrical or more so. There are several ways this can be done. One popular approach would be to assume that the latencies are approximately normally distributed and use a paired t test. Doing so, we find the difference between the load balancing methods is very significant; we reject the null hypothesis at a significance level well below 10^{-5}. (That is, the probability that we'd see such a large apparent difference when there in fact was none is much less than 10^{-5}.)

However, perhaps the assumption of normality is unjustified; the t test may be assigning undue weight to the handful of reports for which the latencies differed greatly, i.e. to the tails of the histogram shown in the figures. (The tails are actually longer and thinner than shown in those figures, since the range was truncated for the sake of clarity; the maximum latency difference between the two schemes is actually 2.9 data periods.)

In order to deal with these doubts, a "robust" test is called for. One likely candidate is Wilcoxon's paired signed rank test. This involves sorting the latency differences into order by their absolute values, and then adding up the positions in this ordering (i.e. the ranks), with the same signs as the differences. That is, we take the sum of the positions occupied by positive differences minus the sum of the positions occupied by negative differences. This weights larger differences more heavily than small ones, but only slightly so out on the tails of the distribution. Moreover, the null distribution of this statistic (i.e. the distribution assuming the null hypothesis holds) is precisely known without needing any assumption of normality, on purely combinatorial grounds. The sign attached to each rank would essentially be determined by a coin flip if the null hypothesis held. Thus the significance level can be assessed with no assumption about the distribution of latencies. Using this test, the significance level turns out somewhat higher than with the t test, but still considerably less than 10^{-5}. In other words, there is no doubt that the scheme proposed in this thesis does actually outperform the recent-information scheme when the larger migration cost is used.

Going back to the case where the smaller migration cost is used, it is important to note that the absolute performance differences between the three dynamic schemes are so slight as to render the question of their statistical "significance" purely academic. The same techniques as described above can be used to argue that the very subtle observed differences between these three schemes are probably reflections of a genuine performance ordering, with our Bayesian scheme intermediate between the other two under these circumstances. However, there is little point in making this case, since the difference is in any case of no consequence.

6.2.3 Increased system loading

To show the impact of increased system loading, figure 6.5 indicates the result of speeding the input data period up from 15 milliseconds to 12.5 milliseconds while retaining the larger process size; the performance of the Bayesian scheme is still superior, and as should be clear from figure 6.6 this difference is quite significant.

6.3 Load estimation errors

Since a major contribution of this thesis is the methodology for using explicit models of the time evolution of load and of the information spreading process to improve system-wide load estimation, it is critical to confirm that this effort actually produces improved load estimates. In particular, we will see that even where our scheme performs worse, its underlying load estimates are still superior. (Regrettably, good estimates do not automatically ensure good performance.)

Rather than do lots of runs with various pseudo-random number generator seeds in order to try to statistically see through the inconsequential differences between runs done with different load-estimation procedures, we have compared data all logged from a single run. This run employed our load-estimation and load-balancing scheme, but also logged the load estimates that would have been obtained using only the current information, and also those that would have been obtained using only the information old enough that the average refinement process had been terminated. This latter option uses the time-series model but without the Bayesian inference to incorporate recent but unreliable information. (The excluded information was most of the information less than one period in age.) The run in question was done with the 15ms data rate and the smaller process state size.

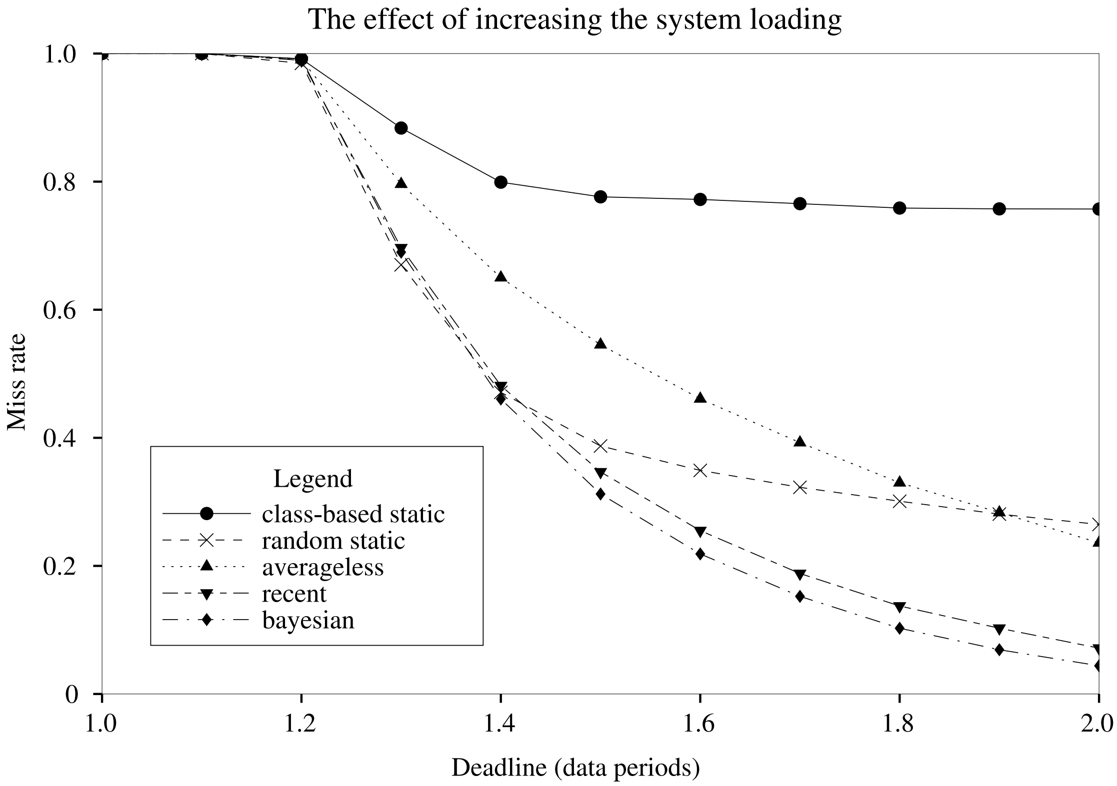

Figure 6.5: The effect of increasing the system loading. This graph again shows the fraction of emitter fix and heading reports missing each of a range of deadlines, shown on the horizontal axis in multiples of the data period, which is now only 12.5ms. The larger process state sizes are still being used.

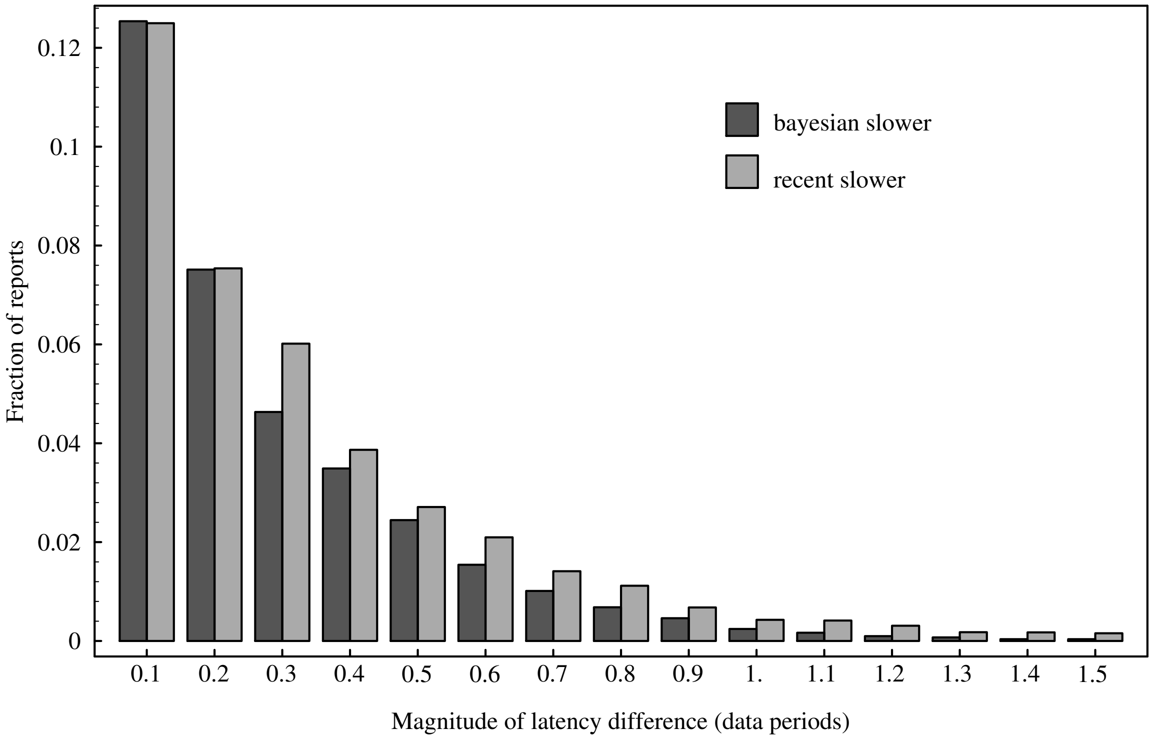

Figure 6.6: High-load latency differences. The Bayesian scheme proposed in this thesis significantly outperforms the recent information scheme under high-load expensive-migration conditions.

The variance of the load estimates around the actual average load was 1.57 for our scheme, 4.64 using only old information, and 3.82 using only current information. Thus both our use of a time-series model to incorporate old information into the estimate and our use of Bayesian inference to also employ recent information are validated. The ratio of the variance using only current information to that with our scheme is about 2.44, rather less than the value of 7.81 we calculated on theoretical grounds in section 4.4. On the other hand, the ratio of the variance with only old information to that with our scheme is 2.97, closer to the 3.30 we derived theoretically.

It is unclear what accounts for these discrepancies between theory and practice; there are enough approximations and simplified models involved in the analysis that there are numerous possible origins for the error. One prime suspect is the fact that the actual implementation of the distributed averaging algorithm results in correlations between the estimation errors for the various time intervals that are not accounted for in our simplified model, as remarked in section 4.4.2. Another likely suspect is the fact that the actual site loads are not normally distributed about the system-wide average, and in fact are not even particularly symmetrically distributed, because the average load is low enough that the fact that no site can have a negative load results in a substantial snubbing off of the distribution in one direction.

6.4 Migration frequency

The primary motivation for producing accurate estimates of the system-wide average load and using that to do global load balancing is that good load balance should be achievable with many fewer object migrations than would be necessary with local load balancing. This is because only objects on overloaded sites will be migrated, and only to underloaded sites. The averageless (local) scheme will in contrast also migrate objects from underloaded sites to yet further underloaded sites, and will migrate objects from overloaded sites to less overloaded sites, even though they will then typically be migrated again. Thus it is clear that the averageless scheme has a weaker standard for when to migrate an object—it only has to find a less loaded site, not one on the opposite side of the system-wide average.

The comparison with the recent-information scheme is less clear. That scheme is also trying to do global load balancing, but using an inferior estimate of the system-wide average load, derived only from recent information from a few sites. If this load estimate were worse in an unbiased fashion, it would appear to result in yet fewer migrations, since it is harder to find sites on opposite sides of an arbitrary dividing line than it is to find sites on opposite sides of the actual average, assuming a relatively symmetrical distribution of loads. However, this assumption of unbiasedness is faulty; among the few sites' loads averaged into the recent-information estimate are the two being compared with that estimate. Therefore, since there are only a few others included, the estimated average is quite likely to fall between the two sites' loads being compared. Thus, we can expect the recent-information scheme to also do more migrations than the proposed Bayesian inference time-series analysis based scheme.

This section presents the empirical evidence supporting these conjectures about migration frequency; the next section examines further the claim that our scheme's more sparing use of migration isn't at the expense of the quality of load balancing. The two sections following that examine the important benefits derived from the reduced number of migrations.

Figure 6.7 shows the distribution of how many times each process migrates for the three dynamic schemes. All three histograms are aggregated over seven runs with different pseudo-random seeds, and all are in the base case, i.e. the data rate is 15 milliseconds, and the process sizes are the original, smaller ones. As can be seen from the figure, the Bayesian time-series analysis based scheme tends to migrate each object fewer times than the averageless scheme, with the recent-information scheme falling in between. Specifically, the average object migrates 1.00 times with the scheme presented in this thesis, 1.29 times with the recent-information scheme, and 1.73 times with the averageless scheme. This translates into correspondingly dramatic differences in the number of objects repeatedly migrated: the number of objects migrated more than once is 60% higher with the averageless method than with the Bayesian one.

The three curves in this figure are all fit extremely well by a geometric probability distribution model, with only the single parameter of that model varying between

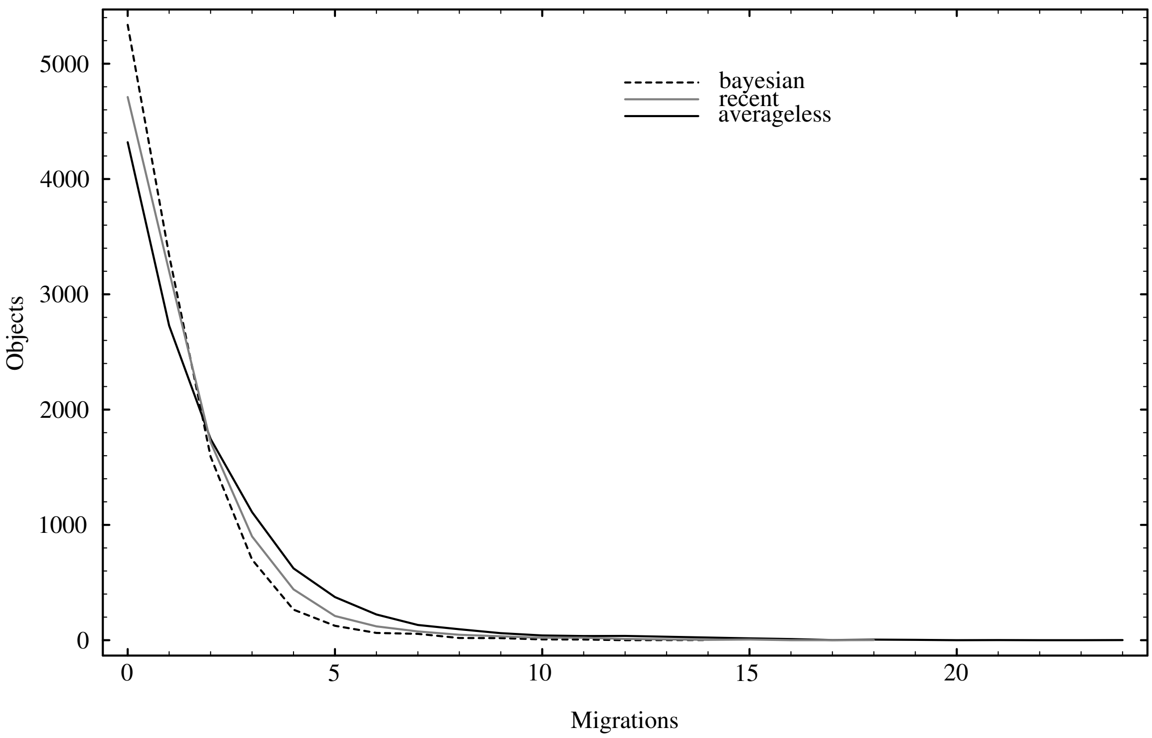

Figure 6.7: Migration frequency. This graph shows on the y-axis how many objects migrated the number of times corresponding on the x-axis.

them. What this means is that there is in each case an essentially fixed probability that a given object located on a given site will migrate off that site. All that varies is this fixed probability; for the averageless scheme, it is 63%, for the recent-information scheme it is 56%, and for our proposed scheme only 50%. This fits well with the remarks at the beginning of this section comparing the three schemes' migration criteria. For example, migrations can occur in the averageless scheme whenever two sites differ in load, while for our scheme the sites must also lie on opposite sides of the system-wide average load. Since this is a more restrictive criterion, it isn't surprising that the probability an object will migrate off a site is lower.

6.5 Load balance

Having seen that our load balancing method does indeed use migrations more sparingly, it remains to show that they are sufficiently strategically chosen as to still adequately balance the load. As can be seen in figure 6.8, although our scheme does not beat the other dynamic migration schemes at load balancing, the resulting load balance is still comparable (and much better than without migration). The very fact that the load balance *isn't* better (in fact, is slightly worse) in this data taken from the large-state case (where performance was better) helps set the stage for the following sections, concerning the benefit of the reduced migration rate. Clearly, something else other than load balance *per se* accounts for the success of our load balancing scheme.

6.6 Overhead

The performance comparisons in section 6.2 reflect not only the performance improvements from better load balance, but also the counteracting costs of the load-balancing mechanism. Those costs are quite substantial, and so it is interesting to examine them in isolation as well. The load balancing algorithm is executed by the operator (housekeeping) processors, so we measured their utilization. Somewhat surprisingly, our Bayesian scheme did not introduce much more operator load than the other dynamic

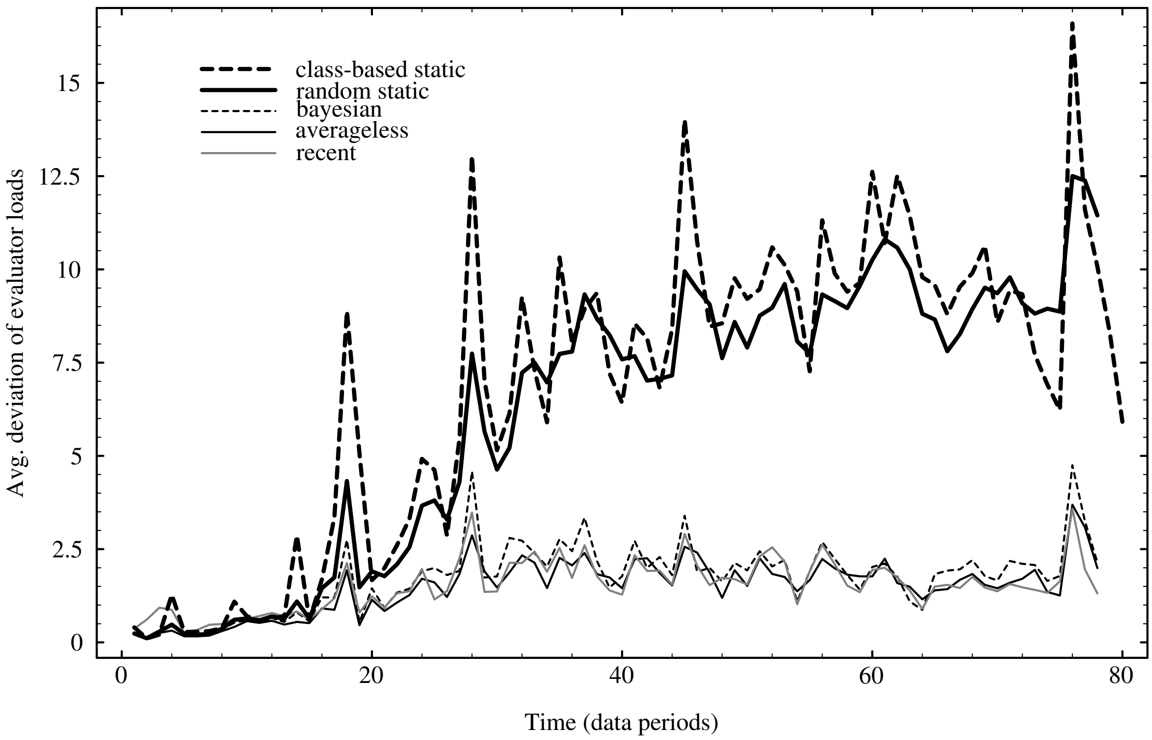

Figure 6.8: Load balance achieved. This graph shows at each time the average of the absolute values of the differences between the individual sites' loads and the system-wide average load.

schemes. With the static random allocation 7% of the operators were busy on the average. The other dynamic schemes increased this to 18% (assuming the larger process state size), while the Bayesian scheme only further increased it to 22%, despite its much more complicated algorithm. (If these numbers all seem so low as to call the existence of a separate housekeeping processor into question, bear in mind that the CARE model does not account for garbage collection, which was a primary purpose for these processors.)

The statistical significance of these differences can again be assessed using Wilcoxon's signed rank test, pairing the utilizations by pseudo-random number generator seeding (i.e., by initial object placement). The difference between the averageless and recent-information schemes is not statistically significant, but the other differences are significant at the .02 level.

The reason why our Bayesian scheme has only slightly higher operator utilization under these circumstances is that the additional load estimation costs of the Bayesian algorithm are partially offset by the savings from performing fewer object migrations and associated communications redirections. Section 6.4 showed that the recent-information scheme performs 29% more migrations than the Bayesian scheme, and the averageless scheme performs a full 74% more migrations than the Bayesian scheme does. (Those numbers were obtained with the smaller process state. With the larger state, the numbers increase to 30% and 82%, but this apparent dependence on state size is statistically insignificant.) Thus the total overhead of the load estimation and object migration remains relatively invariant across the three dynamic load balancing schemes.

With the smaller process state size, the difference in operator utilization is greater: 12% for averageless and 15% for the recent information scheme *vs.* 20% for our scheme. (Again, these differences are significant at the .02 level.) The reason why decreasing the migration cost decreases the overhead of other schemes more is because they do more migrations. Conversely, our scheme is sparing with migrations but has high-overhead load estimation; this load estimation cost isn't reduced by making migrations cheaper.

6.7 The relationship between migrations and latencies

The preceding sections have shown that the proposed load balancing system achieves comparable load balance using many fewer object migrations than the other dynamic schemes, and that this reduction in migrations largely compensates for the increased overhead of the load estimation. However, the reduced number of migrations does far more than that. After all, if that was the entire story, the superior application-level performance observed under large-state conditions would remain unexplained, since there is still slightly higher total overhead with our scheme than with the others. Thus in order to explain superior performance in the face of slightly inferior load balance and slightly greater total overhead, we need to show a further connection between the number of object migrations and the application-level output latencies.

This connection between migrations and latencies is not hard to find. An object that is in transit can not be processing any messages it has already received, nor receiving further messages. Moreover, until such point as the objects sending messages to it have been apprised of its new address, the messages will have to be forwarded from the old site (roughly doubling the communication time). In fact, if the rate of migration is so high as to cause the same object to be repeatedly migrated in quick succession, a forwarding chain may result, in which messages need to be repeatedly forwarded until they catch up with the migrating object.

In order to empirically test this explanation, we can see whether output latencies are correlated with migrations by the relevant objects. Within each separate simulation run, are the reports with longer output latencies those where the objects contributing to the report migrated many times while the report was produced? Any global network congestion, for example, caused by the overall rate of migrations would not show up as a correlation between specific output latencies and the number of migrations of the relevant objects. Conversely, more direct effects of the sort postulated above would show up as a direct correlation.

In order to reduce unrelated sources of variation in output latencies, this experiment used only the emitter heading reports, not the emitter fixes. This improves the chances that a significant correlation between migrations and latencies will be detectable. The objects involved in producing a heading report are a superset of those involved in producing a fix report for the same emitter.

We tested for correlation between the emitter-heading latencies and the number of migrations made by the relevant instances of the classes emitter-manager, emitter-observation, emitter-fix and emitter-heading within the time from one to two data times after the triggering observation. This is the interval during which most of the critical-path processing occurs: there is a one data-time delay to ensure that all the data is input before a report is generated. All four of these object classes are on the critical path for emitter heading reports; one other (the observation reader) is as well in principle, but does not contribute to the variation in latencies at any one time, since a single observation-reader object triggers all reports at a particular time. Further, technical details of our experimental method prevented including the appropriate observation-reader migrations.

We performed this test in each of the runs done with the averageless load-balancing scheme, because that scheme allows the greatest range of migration counts to be observed. Two statistical tests of nonparametric correlation (Spearman's rank correlation r_s and the sum-squared difference of ranks) consistently produced highly significant indications of correlation; all p-values were less than 10^{-9} (that is, there is essentially no chance that the apparent correlation was coincidental).

This indicates that there is a quite real correlation between output latencies and the number of relevant object migrations, but it does not necessarily indicate any causality. In particular, although global effects of the migration rate are eliminated as an explanation, it is possible that the objects that migrated were those on busy sites, and the busyness of those sites, rather than the migrations *per se*, was what lengthened the latencies.

Because of this doubt, we tried the experiment of increasing the process state sizes by a factor of five, and seeing whether the slope of the best-fit latency-vs-migrations line increased. This would provide some indication that the migrations themselves were a significant factor in the latencies. As it happens, the data does bear out this hypothesis: the slope of the best-fit line (computed using the robust method of minimum absolute deviations) increased by a median factor of 5 across the seven pairs of runs.

This result, that the number and size of migrations influences application latencies, helps greatly to explain and motivate the results of section 6.2.

6.8 Migration delay

The time taken by an object migration is an important factor affecting the success of dynamic load balancing mechanisms. The scheme proposed in this thesis is predicated on the assumption that the delay is long enough to make repeated migration undesirable, but short enough that one can afford to do migration at all. In order to experimentally measure the performance of the migration mechanism in our CARE simulation, the various simulation runs were instrumented to record for each migrated object the time when it began being removed from the run queue on the originating site and the time when it was finished being inserted back into the queue of runnable processes, but on the destination site. This includes all overheads of dequeueing and enqueueing the process, encoding the graph-structured state into a linear form and then decoding again, message transmission and reception, network contention, etc.

Figure 6.9 shows that the majority of migrations take a fraction of a millisecond with the smaller state size. The average is .78ms, but that is skewed by the long tail; roughly three quarters of the latencies are under a millisecond. From figure 6.10 you can see that increasing the state size by a factor of five increases the migration delays so that many of them take a few milliseconds and a few take tens of milliseconds. Also, you can see that the mechanisms with more migrations have a larger proportion of migrations that take abnormally long, even though the modal migration delay is still under a millisecond. This increase in long delays shows up quite clearly in the average delay, which ranges from 2.6ms for our scheme, through 2.9ms for the recent information scheme, to a full 4.0ms for the averageless scheme. This is apparently the result of increased network contention caused by the increased number of migrations; the larger object size used in this experiment explains why this contention effect is more evident than in the experiments reported in figure 6.9, even though there as

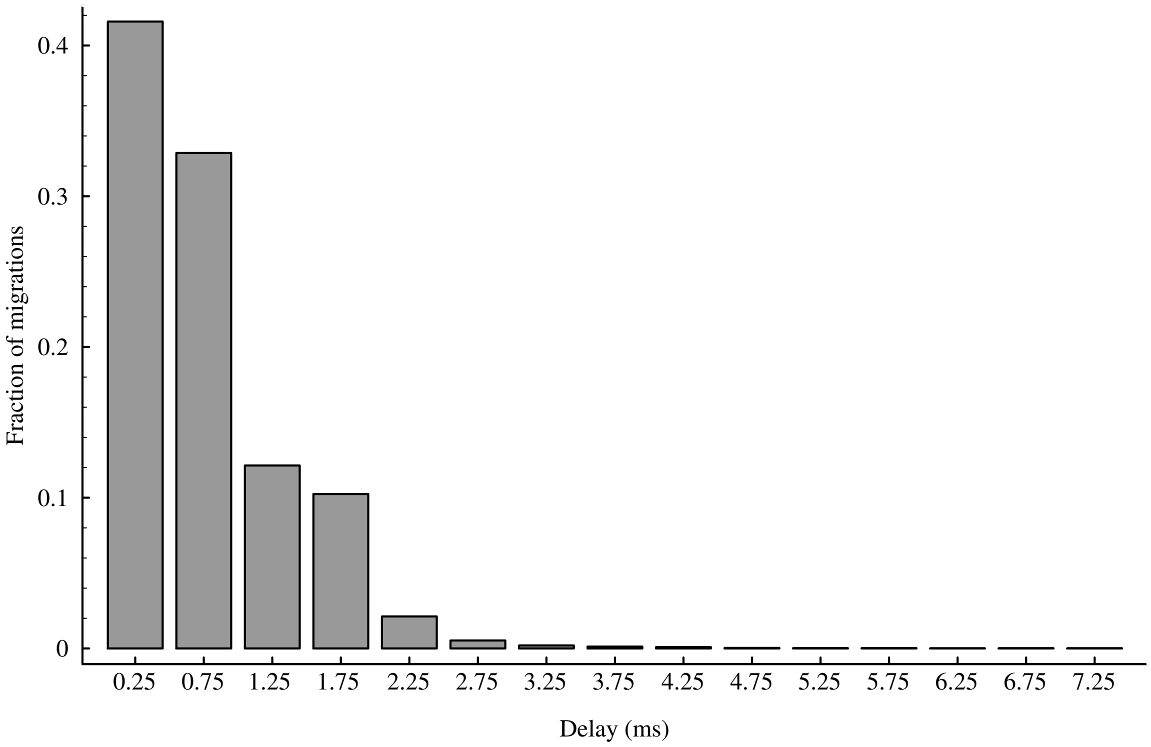

Figure 6.9: Migration delay with small process states. The millisecond values on the x-axis of this histogram are the center-points of .5 ms wide intervals. Each bar shows the fraction of object migrations for which the migration time fell within the indicated interval. For example, the first bar shows that almost 42% of object migrations took less that half a millisecond. This histogram is aggregated over all three dynamic load-balancing schemes; there was only slight variation among them, with the delays increasing somewhat with the number of migrations.

well the number of migrations varies substantially with load-balancing method. Note that the CARE cut-through network responds to contention by delivering a blocked packet to the operator of the site at which the blockage occurred. After waiting in that operator's queue, it is then resent into the network. This introduces a substantial penalty for contention, providing further evidence of the advantage of being sparing in migrations—one of the general strategic assumptions of this thesis.

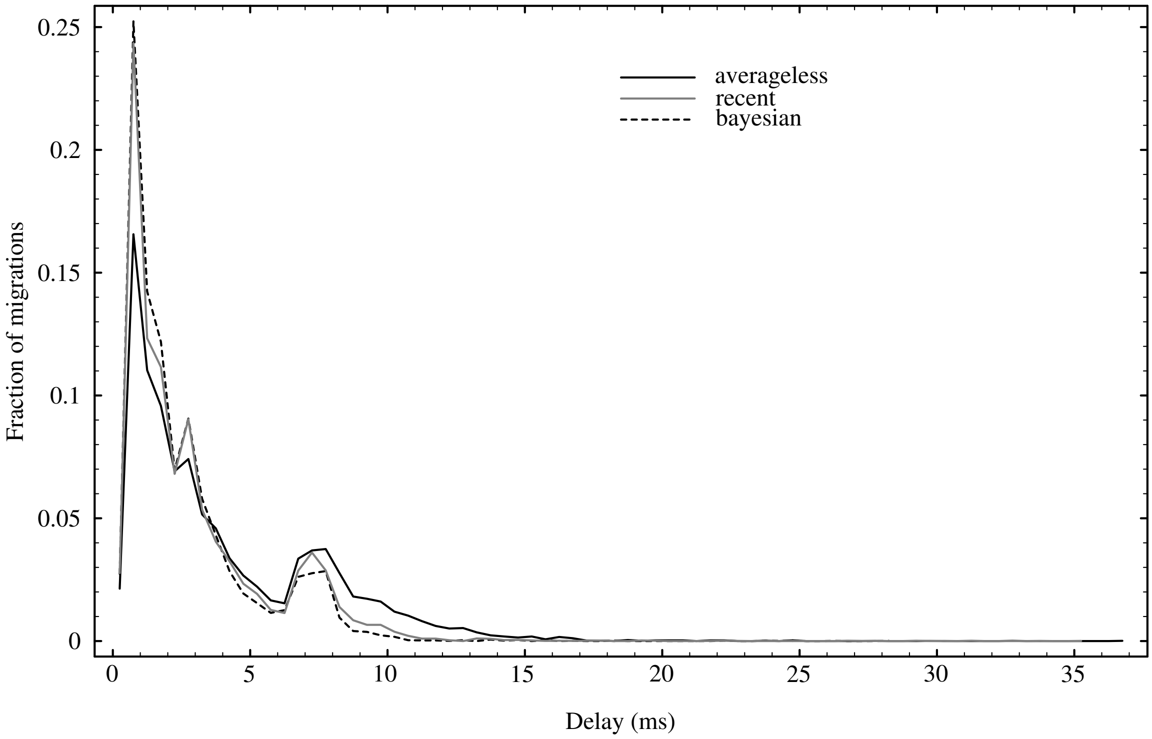

Figure 6.10: Migration delay with larger process states. Each line connects the top-center points of a set of histogram bars, each a half-millisecond wide, showing the fraction of object migrations which took that long. This allows the three histograms to be compared more readily than using traditional bargraphs would. The process state size has been inflated by a factor of five.

6.9 Negotiation delays

One virtue of basing our load-estimation technique on an explicit time-series model of loading is that we could equally easily forecast what the load will be at a future time. In some applications, where the negotiation and execution of a migration are slow compared to changes in system loading, it would be advantageous to forecast the average load a bit into the future, to better reflect the situation when the migration being considered is completed. Is this the case in our experimental example? The evidence suggests not. The primary delay between average estimation and migration completion is the migration itself, the delay of which was discussed in the preceding section. Adding in the "negotiation" time (i.e. the time for the underloaded site to request work from the overloaded site and for the overloaded site to select which objects to migrate) only raises the average delay for the small process states to .83ms, while with the larger state size the average time only goes up to 2.8ms. If you look at the histograms in figures 6.11 and 6.12, you can see that even with the larger state, a large fraction of the time the total of negotiation and migration delays still is less than the 2.5ms load balancing interval width used in these experiments. Thus, given that we are only estimating the load with 2.5ms granularity anyway, it seems that it wouldn't pay to estimate the load even a single interval into the future.

6.10 Synthetic load experiments

In order to better understand the strengths and weaknesses of the proposed load-balancing system, it is worth examining its performance when used to balance the load of an application of a radically different nature than ELINT. This section uses a synthetic application in which the objects are all statically created at the beginning of the run, rather than being dynamically created and destroyed in the course of the run, and are organized into a toroidal grid pattern of nearest-neighbor communication, rather than communicating in arbitrary dynamic patterns.

For an application such as this, there are two obvious object placement policies, which we'll use in addition to random placement. One is to map the toroidal grid

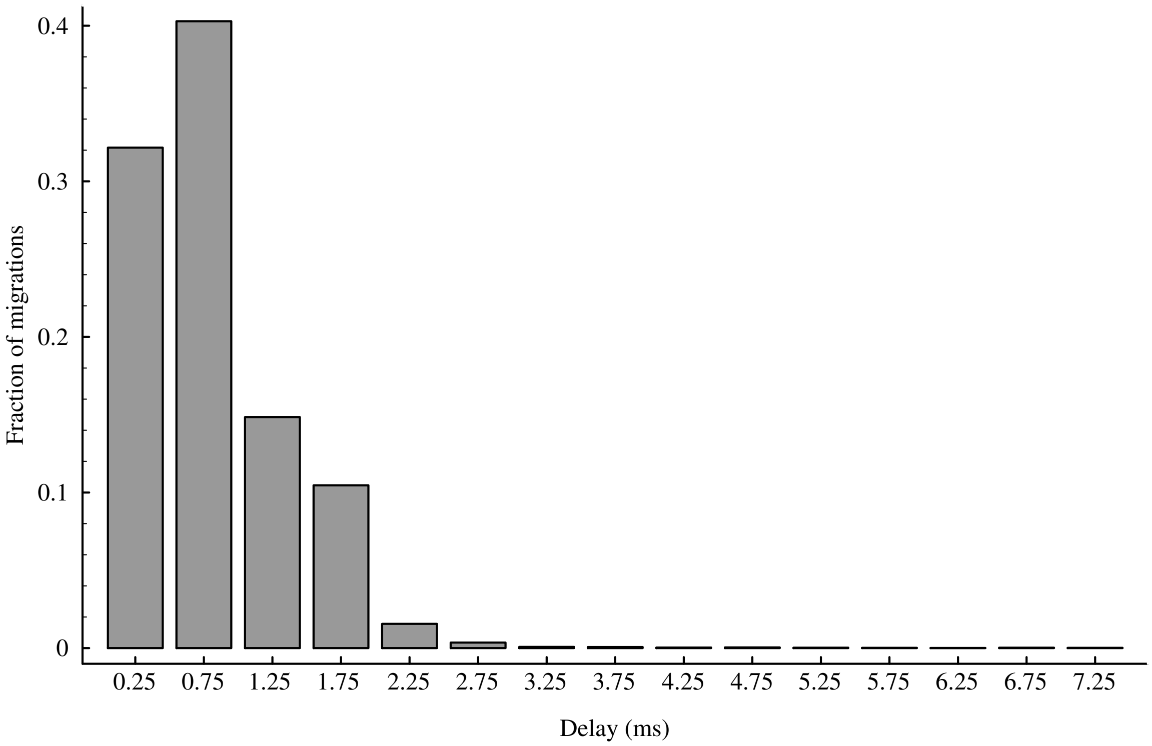

Figure 6.11: Negotiation and migration delay with small process states. The millisecond values on the x-axis of this histogram are again the center-points of .5 ms wide intervals. Each bar shows the fraction of object migrations for which the time from when the underloaded site requested work until the object sent in response was enqueued on the runnable process queue fell within the indicated interval. This histogram is only for the Bayesian scheme, since that is where the question of delay between estimation and completion arises.

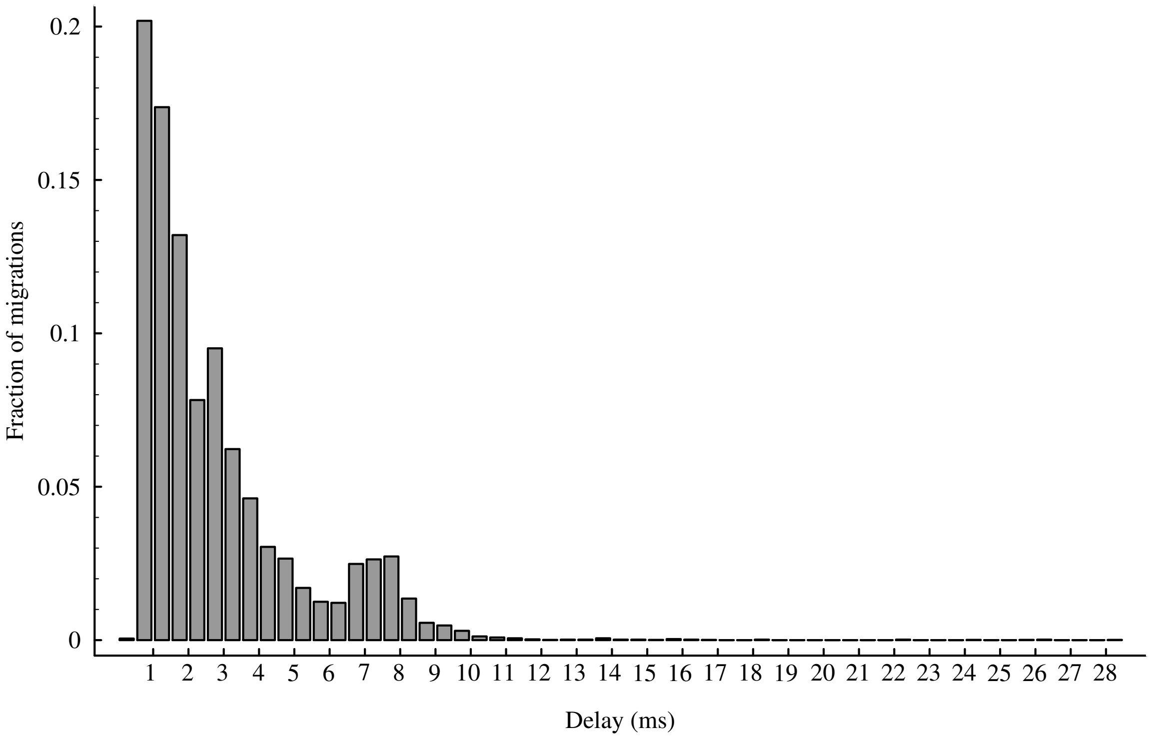

Figure 6.12: Negotiation and migration delay with larger process states. The histogram bars are again one half millisecond wide, even though the x-axis is only marked at integer millisecond points for legibility. The process state size has been inflated by a factor of five. As in the preceding figure, the time intervals measured start when work is requested and end when the received process is runnable.

of objects onto the smaller toroidal grid of processing elements in a *block* fashion, so that several (logically) neighboring objects map onto the same processing element. The other is to do the mapping in a *wrapped* or modular fashion, so that logically neighboring objects are on physically neighboring sites.

The block allocation minimizes communication costs, but can result in poor load balance if the loads of neighboring objects are correlated (as they are in our synthetic application). The wrapped allocation increases the communication cost, though not to the point of random allocation, where in general communications are not only off-site but also to a non-neighboring site. Not only does restricting communication to nearest neighbors reduce the path length the messages have to follow, but it also essentially eliminates the possibility of network congestion. The wrapped object mapping also has the potential for good load balance when loaded objects occur in clusters, since neighboring objects will be on distinct sites.

One major question this section will answer is how these various placement options combine with the four object migration policies described in section 6.1. Can good object migration make up for poor placement? Can a poor choice of migration policy wreck even a good initial placement?

The synthetic application used in these experiments consists of 225 objects, all of a single class, communicating with one another in a 15 by 15 toroidal grid pattern. Each run consists of 25 data intervals, each 25ms long. At the beginning of each data interval, each object is independently selected with .05 probability to serve as an "initiator" of activity, and the selected initiators receive a triggering message containing the current time. (This randomized choice of initiators, as well as all other randomized choices, is pre-computed and used identically in all runs, independent of the choice of object placement and migration strategies.) When an object receives one of these triggering messages, it computes for one or two 100μs periods (randomly decided) and then sends further triggering messages to randomly selected neighbors, containing the same time of activity initiation that was received. Each message is 10 words long; the objects themselves are 200 words in size in one set of experiments, 500 in another. When an object chooses to trigger none of its neighbors, then it outputs a report indicating the time of the original activity initiation that started the chain of triggering messages as well as the current time; the difference between these is taken to be the output latency.

When an object is choosing neighbors to trigger, it effectively flips a weighted coin for each of its neighbors to decide whether or not to trigger it. The weighting of that coin is the same for all the neighbors, but is dependent on the length of the chain of triggering messages from the original initiation of activity. The probability of triggering each of the four neighbors vs. the length of the causal chain so far is as shown in figure 6.13. The three distinct regions are all geometric decreases, but the initial and final regions decrease at a rate of 21% per level of depth in the causal chain while the middle region only decreases 1% per level. The flat region is chosen to lie at the .25 probability point, i.e. the steady state, where each activation triggers an average of one other activation (the flat region continues down to .225 probability). Thus, there is first a period of expanding activity, where each activation triggers several others, then a steady state period where the number of activations remains relatively constant, and then finally a period where the activity dies out. Because the activity spreads only to neighbors, and because the causal chains of activation can turn back on themselves, the activity tends to cluster in compact regions of the object grid.

The decision as to whether to only do one 100μs computation or two before activating neighbors is also done using a similar weighted probabilistic choice with shifting probability. However, rather than being tied to the length of the causal chain, it is based on the number of computations the receiving object has already done for this data period. Further, the probability is increasing, rather than decreasing, and linearly, rather than (piecewise) geometrically. The probability starts at zero each data period and grows linearly, increasing by .01 each time the object is invoked (stopping at 1, of course, even if the object should happen to be invoked more than 100 times in one data period).

Figure 6.14 shows the miss rate vs. threshold for this synthetic load, using the assumption of 200-word objects. It shows all nine combination of the three placement policies (block, wrap, and random) with three migration policies: static, averageless, and "cheat." The "cheat" migration policy is the global load balancing approach

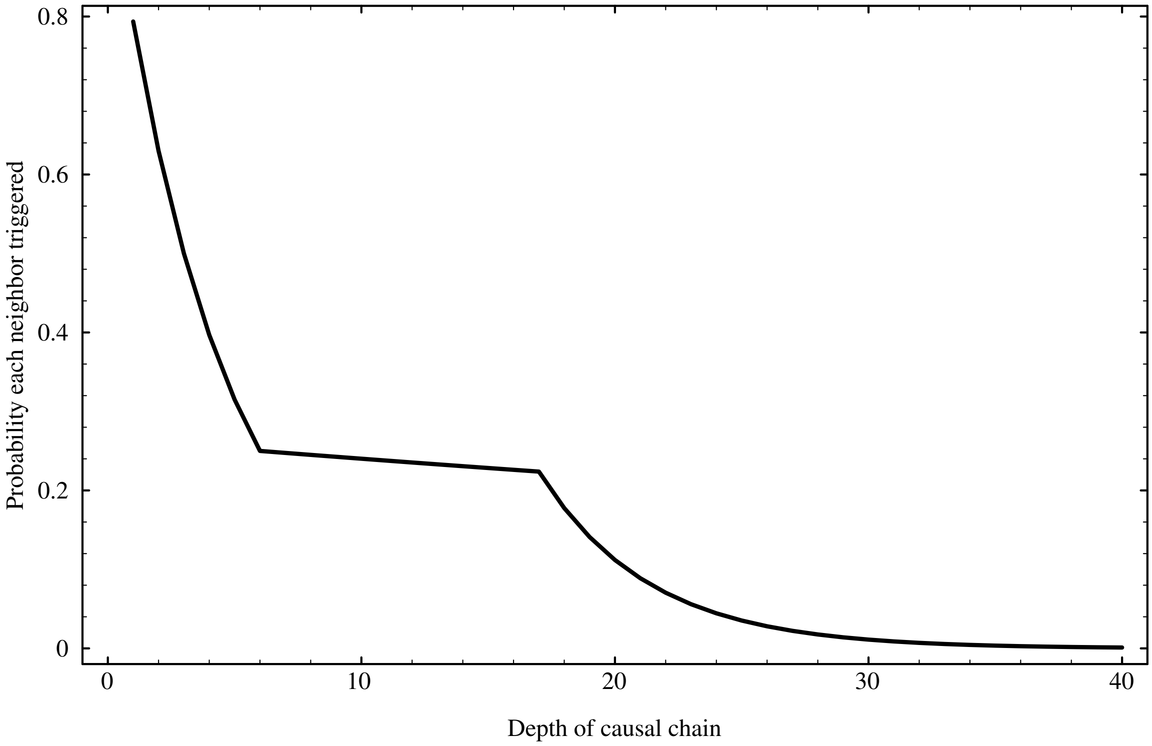

Figure 6.13: Synthetic load trigger probabilities. This shows the probability that an activated object will in turn activate each of its four neighbors, as a function of the number of activation triggerings leading from initiation to this activation. The relatively flat region is around .25, i.e. the steady state where each activation causes an average of one other activation.

studied in this thesis, with the costs of our load estimation method using time-series analysis and Bayesian inference accounted for, but with the actual system-wide average load used rather than the estimate. This is because this synthetic load was not subjected to the ARIMA modeling process. Thus, this evidence reflects the ideal of global load balancing and the costs needed to realize that ideal, but takes for granted that the load can be adequately modeled. In this sense it complements the modeling done for AIRTRAC-DA, where the load was successfully modeled, but no actual load balancing simulations were done.

The best and worst performers in this figure are both static schemes, namely the wrapped and block allocation. The third static scheme, random placement, fares almost as poorly as block placement. Using averageless migration brings all the performance curves together, worsening the performance with wrapped allocation but improving it with random and block allocation, so that the resulting performance works out about the same independent of the initial placement. The results with the cheating version of our proposed scheme are more interesting. The performance with wrapped allocation is hurt less than with averageless migration and the performance with random allocation improved more, so that these two wind up performing comparably to one another and better than the three placements do using averageless migration. However, the performance with block allocation is not improved nearly as much. Presumably the clustering of neighboring objects on sites is so pessimal a load balance as to require more aggressive migration than our scheme can muster.

It is interesting to again see the impact of migration cost on performance, as we did with ELINT, since the smaller number of migrations is a key distinguishing feature of our load balancing scheme. (The various "cheat" runs done with this synthetic load ranged from 707 to 806 migrations, while the averageless runs ranged from 1630 to 1894.) Therefore, figure 6.15 shows the situation when the objects are increased in size to 500 words. (This figure omits block allocation; it seems there would be little reason for anyone to use block allocation for this application. Wrap and random are both plausible, the former because it of its good performance and the latter because it requires little thought of the programmer.) Here the averageless scheme, with its high migration rate, manages to thoroughly ruin the performance of even

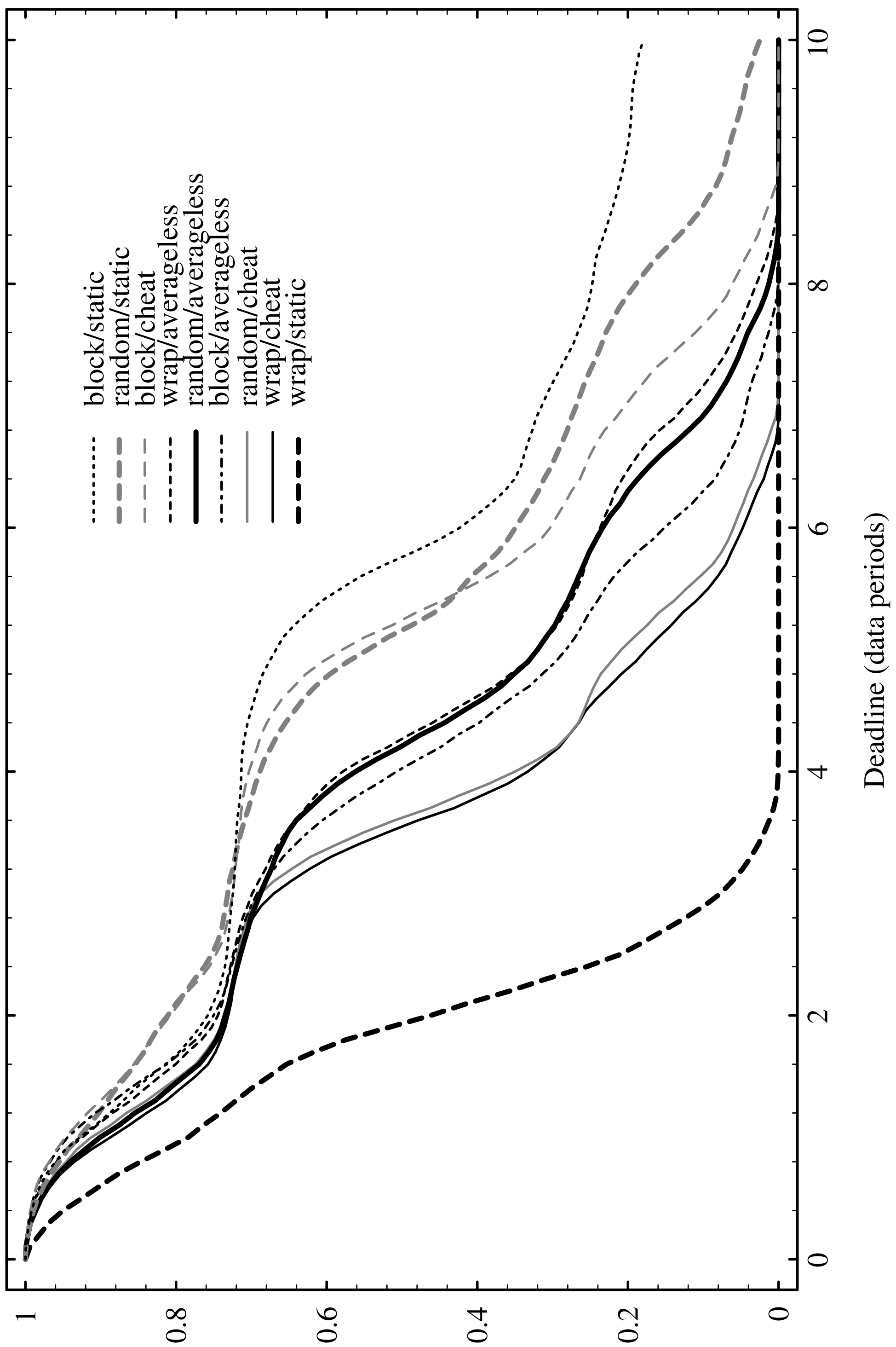

Figure 6.14: Synthetic load, 200 word object size.

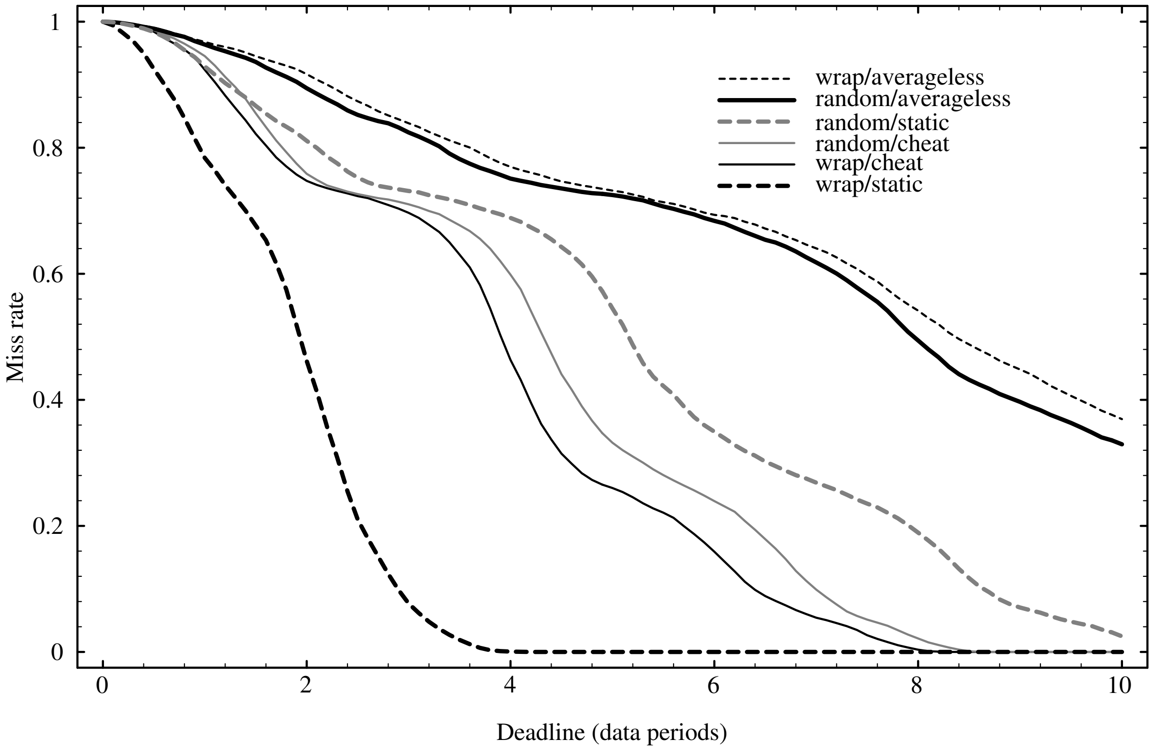

Figure 6.15: Synthetic load, 500 word object size.

wrapped allocation, making it substantially worse than random allocation left alone. The scheme proposed by this thesis, by contrast, shows the potential to improve on random allocation, provided that the average load can be accurately estimated. And, although substantial damage is done to the wrapped allocation performance, it still winds up superior to that with random allocation.

6.11 Summary of performance results

This chapter provided an empirical assessment of our algorithm's experimental implementation from the previous chapter. Rather than simply judging the load balancing method as a "success" or "failure," this chapter provides statistical evidence bearing on each of the individual factors underlying the overall outcome. In particular, we've seen that:

- Our Bayesian inference technique provides load estimates two to three times better than are possible using only recent information or only information old enough to be reliably representative of the whole system.
- As a result, load balance comparable to other dynamic schemes is achieved with 20–40% fewer migrations.
- This decrease in migrations partially offsets the increased overhead, especially in our experiments with higher migration costs, where the total operator load imposed by our scheme is only about one-third higher than that imposed by the other dynamic scheme.
- The decrease in migrations also is salutary for application-level latencies, since the latencies of output reports are strongly associated with the time the relevant objects spend migrating.
- In our experiments with larger object sizes, the decreased number of migrations also reduced the proportion of migrations that took unusually long, presumably by decreasing network congestion.

Chapter 7

Conclusions and Open Questions

We now summarize the problem addressed in the thesis, the methods applied to solve that problem, and the degree to which that solution proved successful. That done, we will conclude with a list of some interesting questions left open for future research.

7.1 The problem

In a modern ensemble machine with a low-latency interconnection network, it is feasible to migrate computational objects between any pair of processing elements. With the opportunities for work transfer wide open, the challenge then becomes to strategically choose which site pairs should actually transfer work.

The introductory chapter noted that in a real-time system, there is strong incentive to reduce the number of object migrations, since each migration extends the latency of any data flowing through the migrated object. One way to reduce migrations might be to sacrifice load balance; however, this would also sabotage the real-time performance goal of consistently short latencies.

Therefore, we are called upon to balance the processing loads but do so with the minimum possible number of object migrations. This has a number of consequences; for example, it motivates our use of the greedy heuristic to select which objects to migrate between a particular pair of sites to transfer a specific amount of load. However, staying with the narrower question of which sites should be chosen, the conclusion we reach is this: work should only be moved from overloaded sites to underloaded sites. Any object migration between two sites on the same side of the system-wide average load is a migration that can be avoided without sacrificing balance.

This criterion—*global* load balancing—is as difficult to implement as it is attractive. In order for a site to identify itself (or a communication partner) as a potential load donor or recipient, it must have available the current average over the processing elements of their loads. This information should ideally be both up-to-date and global in scope—apparently conflicting goals in the large-scale ensembles targeted by this thesis.

The problem that has been addressed by this thesis is to circumvent this impossibility by instead producing a statistical estimate that accurately mirrors the actual current system-wide load using only much more limited information dissemination.

7.2 The solution

Chapter four showed how to use explicit models of the load's variation over time and of the randomized information dissemination in order to produce accurate estimates of the current global load.

The distributed averaging process we examined called for each site to periodically multicast its estimate of the average to a randomly chosen subset of the other sites. Each site recomputes its estimate each interval as the average of those estimates it received in the past interval and its own previous estimate. This produces estimates that improve with age; in particular, their variance decreases geometrically with the number of improvement intervals.

Using a realistic model of how the multicast breadth affects the interval length necessary to stay within a fixed resource budget, we were able to compute an optimal breadth which best trades off interval length against improvement per interval.

In addition to this optimization of the algorithm's parameter, the model of how the estimates improve with age provided one of the two foundations necessary for an appropriate integration of estimates of varying ages. The other needed element, namely a model of how the past loads vary in relevance to the current load, was also presented in the same chapter, in the form of ARIMA time-series modelling. Multiplicative IMA models were used to capture the statistical structure in the Airtrac and ELINT loads.

These two models in hand, a multivariate Bayesian inference allowed an optimal set of weights to be calculated for estimating the current load from the observed past loads. Analysis suggested that this could achieve considerably better results than using either only data old enough to be reliable or using only recent data.

7.3 Outcome and open questions

Although the formulation and solution of the load estimation problem is a major contribution in its own right, it is equally important to consider the empirical results achieved using these techniques. Chapter six presented these results, obtained using the implementation from chapter five.

One of the strongest experimental results is that the use of global load balancing (i.e. estimates of the system-wide average load) significantly reduces the number of object migrations without substantially detracting from load balance.

Another clear result is that the number of object migrations is strongly correlated to the application-level output latencies. Moreover, as migrations become more costly, the impact on latencies is correspondingly increased.

These two experimental results combine to validate the basic premise of the thesis: for the class of real-time systems under consideration, global load balancing is an attractive approach to achieving consistently low latencies, because it can balance the load without incurring the costs of excess object migrations.

Beyond demonstrating the virtues of global load balancing as an objective, the experimental evidence also bears on the question of whether the proposed techniques can actually provide the inexpensive, accurate system-wide load estimates needed to achieve that objective. The results regarding accuracy of the load estimates are encouraging; although the improvement over using only recent information was not as large as had been predicted, it was still substantial, as was the improvement over only using information old enough to be representative of the entire system. (Moreover, the improvement over using only recent information is the area where scaling up to a more realistic ensemble size is most likely to help.)

The proposed global load-balancing technique had its weakest showing in the area of overhead costs. The load estimation costs were so high that even with the larger of the two migration costs used in the experiments, the dramatically smaller number of migrations didn't fully compensate for the overhead. It is this high overhead that most calls into question the wisdom of the proposed technique, especially in systems with low object migration costs.

However, better application-level performance was achieved even with a slight net increase in overhead, because the number of migrations impacts the output latencies other than just by influencing overall overhead. Since this relation between migrations and latencies is stronger with larger migration costs and the net overhead difference is also smaller in that case, it is not surprising that the experimental results showed an improvement in performance only with the more expensive migrations.

The synthetic load experiments suggest that global load balancing can also be useful to compensate for lack of a good static mapping in highly regular computations, again especially when migrations are relatively expensive. However, not surprisingly, when a good static mapping is available, it is preferable to any kind of dynamic object migration.

Assessing the overall outcome of this experimentation, the proposed techniques for global load estimation and global load balancing were shown to be fundamentally promising and to largely achieve their intended objectives. However, the net improvement in application-level performance was quite small, and even that was only possible when object migration delays were increased to several times the typical method execution times. In a system closely comparable to the experimental one, therefore, the proposed techniques do not seem to warrant their high complexity for practical application.

However, in systems with higher penalties for migration and in which broadly representative data on system loading is not quickly available, our approach would be more attractive. It is worth noting that in our experiments the close competitor for performance is the recent-information scheme, which is precisely the one which might be expected to suffer the most from scale-up. (This is one of the most important of the open questions listed below.)

Another important area for further research would be the application of our basic load estimation ideas to a coarser-grained application running on a distributed network of separate workstation computers. This sort of environment might well have costly enough migrations to warrant a global balancing approach, while having sufficiently slow global information dissemination to warrant using our time-series model. The basic principle demonstrated in this thesis, which may be useful in this or other domains, is that the time evolution of load can have exploitable structure, making it possible to judge potential migrations by a strict, global standard in the absence of up-to-date global information.

Bibliography

[1] G. A. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[2] Noga Alon, Amnon Barak, and Udi Manber. On disseminating information reliably without broadcasting. In *The Seventh International Conference on Distributed Computing Systems*, pages 74–81, September 1987.

[3] Ramune Arlauskas. iPSC/2 system: A second generation hypercube. In *The Third Conference on Hypercube Concurrent Computers and Applications*, pages 38–42. ACM Press, January 1988.

[4] Yeshayahu Artsy, Hung-Yang Chang, and Raphael Finkel. Processes migrate in Charlotte. Technical Report 655, Computer Sciences Department, University of Wisconsin–Madison, August 1986.

[5] William C. Athas. Fine grained concurrent computations. Technical Report 5242:TR:87, California Institute of Technology Department of Computer Science, May 1987. Ph.D. thesis.

[6] Amnon Barak and Zvi Drezner. Distributed algorithm for the average load of a multicomputer. Technical Report CRL-TR-17-84, Computing Research Laboratory, University of Michigan, March 1984.

[7] Amnon Barak and Yoram Kornatzky. Design principles of operating systems for large scale multicomputers. Research Report RC 13220, IBM T. J. Watson Research Center, October 1987.

[8] Amnon Barak and Amnon Shiloh. A distributed load-balancing policy for a multicomputer. *Software—Practice and Experience*, 15(9):901–913, September 1985.

[9] Katherine M. Baumgartner and Benjamin W. Wah. Load balancing protocols on a local computer system with a multiaccess network. In *Proceedings of the 1987 International Conference on Parallel Processing*, pages 851–858. The Pennsylvania State University Press, 1987.

[10] Shekhar Borkar et al. iWarp: An integrated solution to high-speed parallel computing. In *Supercomputing '88*, November 1988.

[11] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day Inc., 1976.

[12] Harold D. Brown, Eric Schoen, and Bruce A. Delagi. An experiment in knowledge-based signal understanding using parallel architectures. Technical Report STAN-CS-86-1136, Department of Computer Science, Stanford University, October 1986.

[13] Gregory T. Byrd, Nakul P. Saraiya, and Bruce A. Delagi. Multicast communication in multiprocessor systems. In *International Conference on Parallel Processing*, volume I, pages 196–200. The Pennsylvania State University Press, August 1989.

[14] Clemens H. Cap and Volker Strumpen. The Parform—a high performance platform for parallel computing in a distributed workstation environment. Technical Report ifi-92.07, Institut für Informatik, Universität Zürich, June 1992.

[15] Hung-Yang Chang. Dynamic scheduling algorithms for distributed soft real-time systems. Technical Report 728, Computer Sciences Department, University of Wisconsin–Madison, 1987. Ph.D. thesis.

[16] Timothy C. K. Chou and Jacob A. Abraham. Distributed control of computer systems. *IEEE Transactions on Computers*, C-35(6):564–567, June 1986.

[17] Shyamal Chowdhury. The greedy load sharing algorithm. Technical Report 87-23, Department of Computer Science, The University of Arizona, August 1987.

[18] William J. Dally. Wire-efficient VLSI multiprocessor communications networks. In *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*, pages 391–415. The MIT Press, 1987.

[19] Bruce A. Delagi, Nakul Saraiya, Greg Byrd, and Sayuri Nishimura. CARE user manual. Technical Report KSL-88-53, Knowledge Systems Laboratory, Computer Science Department, Stanford University, September 1990. Version 0 (for Release 0).

[20] Bruce A. Delagi and Nakul P. Saraiya. ELINT in LAMINA: Application of a concurrent object language. *SIGPLAN Notices*, 24(4):194–196, April 1989.

[21] Bruce A. Delagi, Nakul P. Saraiya, and Gregory T. Byrd. LAMINA: CARE applications interface. Technical Report KSL-86-67, Knowledge Systems Laboratory, Computer Science Department, Stanford University, November 1987.

[22] Zvi Drezner and Amnon Barak. A probabilistic algorithm for scattering information in a multicomputer system. Technical Report CRL-TR-15-84, Computing Research Laboratory, University of Michigan, March 1984.

[23] Zvi Drezner and Amnon Barak. An asynchronous algorithm for scattering information between the active nodes of a multicomputer system. *Journal of Parallel and Distributed Computing*, 3(3):344–351, September 1986.

[24] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogenous distributed systems. *IEEE Transactions on Software Engineering*, SE-12(5):662–675, May 1986.

[25] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. A comparison of receiver-initiated and sender-initiated adaptive load sharing. *Performance Evaluation*, 6(1):53–68, March 1986.

[26] D. F. Ferguson, Y. Yemini, and C. N. Nikolaou. Microeconomic models for resource sharing in a multicomputer. Research Report RC 12792, IBM T. J. Watson Research Center, May 1987.

[27] Donald Ferguson, Yechiam Yemini, and Christos Nikolaou. Microeconomic algorithms for load balancing in distributed systems. In *International Conference on Distributed Computer Systems*, pages 491–499. IEEE, 1988.

[28] Robert Joseph Fowler. Decentralized object finding using forwarding addresses. Technical Report 85-12-1, Department of Computer Science, University of Washington, December 1985. Ph.D. thesis.

[29] Dirk C. Grunwald. Circuit switched multicomputers and heuristic load placement. Technical Report UIUCDCS-R-89-1514, Department of Computer Science, University of Illinois at Urbana-Champaign, September 1989. Ph.D. thesis.

[30] Max Hailperin. Load balancing for massively-parallel soft-real-time systems. Technical Report STAN-CS-88-1222, Department of Computer Science, Stanford University, August 1988. Also appeared in condensed form in *Frontiers '88: The Second Symposium on the Frontiers of Massively Parallel Computation*.

[31] Robert H. Halstead, Jr. and Stephen A. Ward. The MuNet: A scalable decentralized architecture for parallel computation. In *Proc. 7th Annual Symposium on Computer Architecture*, pages 139–145, May 1980.

[32] J. P. Huang, J. C. Shih, and T. L. Machleit. Load balancing of a distributed radar system test driver. In *Proceedings Real-Time Systems Symposium*, pages 133–140, December 1982.

[33] Bernardo A. Huberman and Tad Hogg. The behavior of computational ecologies. In B. A. Huberman, editor, *The Ecology of Computation*, pages 77–115. Elsevier Science Publishers B.V. (North-Holland), 1988.

[34] Paul Hudak and Benjamin Goldberg. Experiments in diffused combinator reduction. In *1984 ACM Symposium on Lisp and Functional Programming*, pages 167–176, August 1984.

[35] Jie-Yong Juang and Benjamin W. Wah. Load balancing and ordered selections in a computer system with multiple contention buses. *Journal of Parallel and Distributed Computing*, 7(3):391–415, December 1989.

[36] Eric Jul. Object mobility in a distributed object-oriented system. Technical Report 88-12-06, Department of Computer Science, University of Washington, December 1988. Ph.D. thesis.

[37] L. V. Kalé. Comparing the performance of two dynamic load distribution methods. In *Proceedings of the 1988 International Conference on Parallel Processing*, volume I, pages 8–12. The Pennsylvania State University Press, August 1988.

[38] Richard Korry. A load sharing algorithm for a workstation environment. Technical Report 86-10-03, Department of Computer Science, University of Washington, October 1986. MS thesis.

[39] Phillip Krueger and Raphael Finkel. An adaptive load balancing algorithm for a multicomputer. Technical Report 539, Computer Sciences Department, University of Wisconsin–Madison, April 1984.

[40] Phillip Krueger and Miron Livny. Load balancing, load sharing and performance in distributed systems. Technical Report 700, Computer Sciences Department, University of Wisconsin–Madison, August 1987.

[41] Phillip E. Krueger. Distributed scheduling for a changing environment. Technical Report 780, Computer Sciences Department, University of Wisconsin–Madison, June 1988. Ph.D. thesis.

[42] James F. Kurose and Renu Chipalkatti. Load sharing in soft real-time distributed computer systems. *IEEE Transactions on Computers*, C-36(8):993–1000, August 1987.

[43] Frank C. H. Lin and Robert M. Keller. Gradient model: A demand-driven load balancing scheme. In *The Sixth International Conference on Distributed Computing Systems*, pages 329–336, 1986.

[44] Samprakash Majumdar and Michael L. Green. A distributed real time resource manager. In *Distributed Data Acquisition, Computing, and Control Symposium*, pages 185–193, December 1980.

[45] Russell Nakano and Masafumi Minami. Experiments with a knowledge-based system on a multiprocessor. Technical Report STAN-CS-87-1188, Department of Computer Science, Stanford University, October 1987.

[46] David M. Nicol. Mapping a battlefield simulation onto message-passing parallel architectures. In *Distributed Simulation 1988*, pages 141–146. The Society for Computer Simulation, 1988.

[47] David M. Nicol and Jr. Paul F. Reynolds. Optimal dynamic remapping of data parallel computations. *IEEE Transactions on Computers*, 39(2):206–219, February 1990.

[48] David M. Nicol and Joel H. Saltz. Principles for problem aggregation and assignment in medium scale multiprocessors. Technical Report 87-39, Institute for Computer Applications in Science and Engineering, September 1987.

[49] Joseph Carlo Pasquale. Intelligent decentralized control in large distributed computer systems. Technical Report UCB/CSD 86/422, Computer Science Division (EECS), University of Califonia, Berkeley, April 1988. Ph.D. thesis.

[50] Krithi Ramamritham, John A. Stankovic, and Wei Zhao. Distributed scheduling of tasks with deadlines and resource requirements. Technical Report 88-92, Department of Computer and Information Science, University of Massachusetts at Amherst, October 1988.

[51] T. M. Ravi and David Jefferson. A basic protocol for routing messages to migrating processes. In *Proceedings of the 1988 International Conference on Parallel Processing*, volume II, pages 188–197. The Pennsylvania State University Press, August 1988.

[52] Thirumalai Muppur Ravi. Routing messages to migrating processes in large distributed systems. Technical Report CSD-890049, University of California at Los Angeles Computer Science Department, August 1989. Ph.D. thesis.

[53] James Rice. The Advanced Architectures Project. Technical Report KSL 88-71, Knowledge Systems Laboratory, Computer Science Department, Stanford University, December 1988.

[54] Nakul P. Saraiya, Bruce A. Delagi, and Sayuri Nishimura. Design and performance evaluation of a parallel report integration system. Technical Report KSL-89-16, Knowledge Systems Laboratory, Computer Science Department, Stanford University, April 1989. A condensed version of this paper was published as [20].

[55] Wei Shu and L. V. Kalé. Dynamic scheduling of medium-grained processes on multicomputers. Technical Report UIUCDCS-R-89-1528, Department of Computer Science, University of Illinois at Urbana-Champaign, May 1989.

[56] John A. Stankovic. An application of Bayesian decision theory to decentralized control of job scheduling. *IEEE Transactions on Computers*, C-34(2):117–130, February 1985.

[57] Marvin M. Theimer and Keith A. Lantz. Finding idle machines in a workstation-based distributed system. *IEEE Transactions on Software Engineering*, 15(11):1444–1458, November 1989.

[58] Brian K. Totty. An operating environment for the Jellybean Machine. Bachelor's thesis, Massachusetts Institute of Technology, May 1988.

[59] Minoru Uehara and Mario Tokoro. An adaptive load balancing method in the computational field model. *OOPS Messenger*, 2(2):109–113, April 1991. Proceedings of the ECOOP-OOPSLA Workshop on Object-based Concurrent Programming.

[60] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and Scott Stornetta. Spawn: A distributed computational economy. Technical report, Dyamics of Computation Group, Xerox Palo Alto Research Center, 1989.

[61] Yung-Terng Wang and Robert J. T. Morris. Load sharing in distributed systems. *IEEE Transactions on Computers*, C-34(3):204–217, March 1985.

[62] Mark Williams, Harold Brown, and Terry Barnes. TRICERO design description. Technical Report ESL-NS539, ESL, Inc., May 1984.

[63] Songnian Zhou. Performance studies of dynamic load balancing in distributed systems. Technical Report UCB/CSD 87/376, Computer Science Division (EECS), University of Califonia, Berkeley, October 1987. Ph.D. thesis.

[64] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: A load sharing facility for large, heterogeneous distributed computer systems. Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, April 1992.

[65] Glenn Zorpette. The power of parallelism. *IEEE Spectrum*, 29(9):28–33, September 1992.