PARALLEL I/O- AND COMMUNICATION-SENSITIVE SCHEDULING ON HIGH-PERFORMANCE PARALLEL COMPUTERS

by

JENS MACHE

A DISSERTATION

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

March 1999

"Parallel I/O- and Communication-Sensitive Scheduling on High-Performance Parallel Computers," a dissertation prepared by Jens Mache in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science. This dissertation has been approved and accepted by:

Virginia M. Lo

Dr. Virginia M. Lo, Chair of the Examining Committee

March 8, 1999

Date

Committee in charge:

Dr. Virginia M. Lo, Chair
Dr. Sharad Garg
Dr. Richard Koch
Dr. Marilynn Livingston
Dr. Allen Malony
Dr. Andrzej Proskurowski

Accepted by:

Dean of the Graduate School

An Abstract of the Dissertation of

Jens Mache for the degree of Doctor of Philosophy

in the Department of Computer and Information Science

to be taken March 1999

Title: PARALLEL I/O- AND COMMUNICATION-SENSITIVE SCHEDULING ON HIGH-PERFORMANCE PARALLEL COMPUTERS

Approved: Virginia M. Lo

Dr. Virginia M. Lo

A serious bottleneck to high-performance parallel computing is the high cost of data transfer. As communication and I/O traffic on the links in the interconnection network increases, *network contention* becomes a critical problem, drastically reducing effective data throughput. Minimizing network contention is crucial in order to achieve fast job response times and high system throughput.

One factor that affects network contention is the resource management issue of *processor allocation*, the assignment of a set of processors to each scheduled job. Previous processor allocation strategies have essentially ignored the I/O and communication demands of parallel applications and the resulting network contention. Our analysis shows that the *spatial layout* of the compute nodes and the I/O nodes

in relation to each other within the interconnection network topology is a key factor that affects network contention. Based on the results of this analysis, we design and test new processor allocation strategies that minimize network contention by being sensitive to spatial layout and its effect on communication and parallel I/O.

Our analysis is based on analytic modeling and on simulations driven by synthetic workloads and realistic workload traces captured at supercomputing sites. We analyze and minimize network contention in three different situations. First, we concentrate on *communication* intensive jobs. We analyze inter-job link contention due to communication among compute nodes, and we design a new strategy that allocates each job as compactly as possible. Second, we concentrate on *parallel I/O* intensive jobs. We analyze traffic hotspots due to data transfer between compute nodes and I/O nodes, and we design new strategies that optimize the shape and location of jobs relative to the I/O nodes. Strategies that are optimal for parallel I/O often conflict with strategies that are optimal for communication. Thus as a final step, we design an integrated allocation strategy that accommodates workloads that are both communication intensive and parallel I/O intensive. Our new strategies are successful in improving both average job response times and system throughput, and thus make a contribution towards efficient resource management of teraflops-scale computing systems.

CURRICULUM VITA

NAME OF AUTHOR: Jens Mache

PLACE OF BIRTH: Karlsruhe, Germany

DATE OF BIRTH: April 10, 1970

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon
Southern Oregon University
Universität Karlsruhe, Germany

DEGREES AWARDED:

Doctor of Philosophy in Computer Science, 1999, University of Oregon
Master of Science in Mathematical and Computer Science, 1994, Southern Oregon University
Vordiplom in Computer Science, 1992, Universität Karlsruhe

PUBLICATIONS:

Lo, Virginia, Mache, Jens, and Windisch, Kurt. A Comparative Study of Real Workload Traces and Synthetic Workload Models for Parallel Job Scheduling. In *Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '98*, pp. 1–16, 1998.

Mache, Jens, and Lo, Virginia. The Effects of Dispersal on Message-Passing Contention in Processor Allocation Strategies. In *Proceedings of the 3rd Joint Conference on Information Sciences*, Volume 3, Sessions on Parallel and Distributed Processing, pp. 223–226, 1997.

Mache, Jens, Lo, Virginia, and Windisch, Kurt. Minimizing Message-Passing Contention in Fragmentation-Free Processor Allocation. In *Proceedings of the 10th International Conference on Parallel and Distributed Computing Systems*, pp. 120–124, 1997.

Mache, Jens, Lo, Virginia, Livingston, Marilynn, and Garg, Sharad. The Impact of Spatial Layout of Jobs on Parallel I/O Performance. In *Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems, FCRC '99*, 1999.

ACKNOWLEDGEMENTS

I am very grateful to my advisor, Dr. Virginia Lo. She is an exceptional scholar and mentor. She persistently pushes for highest quality.

I would like to thank the members of my committee, Richard Koch, Allen Malony and Andrzej Proskurowski, and especially Sharad Garg and Marilynn Livingston.

Thanks to the people who made a big difference in my life: my parents, Gudrun and Holm-Rainer Mache, my sister, Daniela, as well as Dirk Notheis, Lee Hill, and Ursula Schwantag.

Thanks to my friends and acquaintances around the world: Timo Ansbach, Vimala Appayya, Achim Baumgartner, Sanda Bilcar, Sue Blair, Jerry and Katharine Boness, Chris Bornman, Jana Brocking, Thomas and Michaela Bundschuh, Terry and Anne Carter, Aida Chouchane, Michael Curry, Eck Doerry, Srinath Duvuru, Enrique Franco, Arne Frick, Susanne Friese, Patrick Gee, Michael Haag, Chris Harrop, Oliver Hauck, Hans Heidle, Birgit and Rob Hindman, Chris Hundhausen, Tobias Kässer, Brendan Kane, Jan Kratochvil, Thomas Längle, Mark Lawson, Benjamin Lu, Jayne Miller, Bernd Mohr, Marcus Munk, Bill Nitzberg, Diane Notheis, Silja Pamme, Ferenc Rakoczi, Yolanda Reimer, Steffen Rodig, Martin Saar, Gunnar Sacher, Bethina Schmitt, Gordon Seitz, Miley Semmelroth, Sameer Shende, Oliver Stern, Craig Thornley, Tammi Vincik, and Kurt Windisch.

DEDICATION

To Ursula

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

Figure Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

> The biggest obstacle to effective use of teraflops-scale computing systems is getting data into, out of and around such systems fast enough to avoid severe bottlenecks. Paul Messina [48]

Today's high-performance parallel computers have the potential to achieve processing powers of more than one trillion floating point operations per second. However, as expressed in the above quote, the cost of data transfer needed to support communication and I/O has become a key limiting factor in the quest for teraflops performance. The data and I/O demands of applications in many fields – especially in computational science, visualization, database, and multimedia – are increasing at an unprecedented pace. Whereas the world-wide web (WWW) today contains less than one terabyte of HTML, satellite image databases grow by one terabyte in a few hours. The Sloan Sky Survey operates on 40 terabytes of data, and NASA's Earth Observing System Data and Information System (EOSDIS) will operate soon on 15 petabytes of data. Other extremely data-intensive projects include digital libraries, deciphering nature's DNA code or modeling the atmosphere and the oceans.

These vast amounts of data must be moved between compute nodes and I/O nodes, and among compute nodes as well. A crucial challenge is effective and efficient use of the shared medium for data transfer: the interconnection network.

As the traffic on the links in the network increases, *network contention* becomes a critical problem, drastically reducing effective data and I/O throughput rates. Network contention occurs if several data transfers want to use the same network link at the same time (see Figure 1).

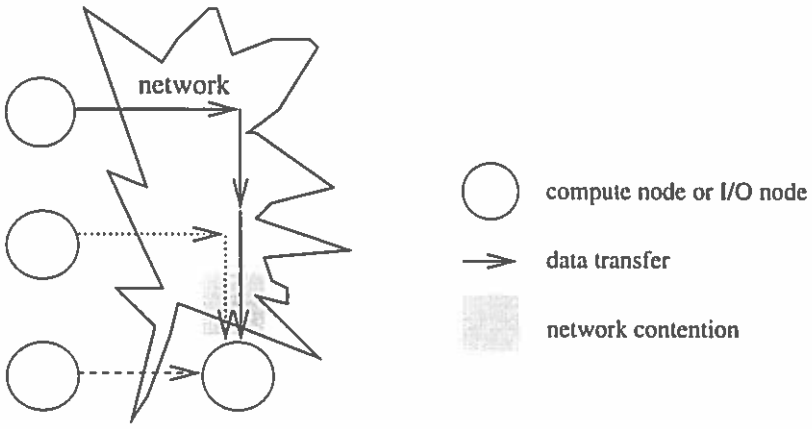


FIGURE 1. Network Contention

Network contention has been reported to cause significant delays in communication traffic [9, 6, 33, 40, 42] and to degrade I/O performance [5, 10, 20, 28]. For example, measurements on a Symult 2010, an Intel DELTA and an Intel Paragon showed that contention for network links increased the communication time per message exponentially as the communication-to-computation ratio increases [9]. As another example, I/O measurements on a TFLOPS machine with up to nine I/O nodes and several hundred processor nodes at Intel [22, 24] showed that certain placements of compute nodes and I/O nodes saturated the bandwidth of some network links, resulting in serious parallel I/O performance degradation [23].

Minimizing network contention is crucial in order to achieve fast job response times and high system throughput. A large body of research has focused on techniques to minimize network contention, ranging from theoretical work in mapping

and embedding, to the implementation of communication and I/O libraries, to the design of specialized hardware for efficient data transfer.

The approach we take focuses on the role played by the resource management task known as *processor allocation.* In a typical message-passing parallel machine, jobs arrive, are scheduled for execution on one or more compute nodes, execute, and depart the system. Processor allocation is one part of the scheduling procedure in which a set of processors is selected from the pool of available processors for assignment to the job. Previous research has shown that the spatial layout of nodes involved in data transfer within the network topology, coupled with the underlying routing schema, directly affects the network contention levels. Yet, no one has conducted a systematic analysis of the effects that the processor allocation strategy, through its spatial layout decisions, has on network contention, and little effort has been made to tune processor allocation strategies for communication and I/O-intensive workloads.

The goal of our research is the development of communication and I/O sensitive processor allocation strategies for high-performance message-passing architectures. To this end, we investigate the effect of *spatial layout* of compute nodes and I/O nodes on network contention. Spatial layout includes location, shape and compactness of allocated nodes. We focus on contention arising from two distinct arenas: (1) communication traffic among compute nodes and (2) I/O traffic between compute nodes and I/O nodes. For each domain, we develop a model and appropriate metrics for measuring contention. Through graph theoretic analysis and dynamic simulation, we analyze the relationship between spatial layout, network contention, and throughput. We then apply what we have learned to the

design of several new processor allocation strategies, and evaluate their performance through simulation using both stochastic and real workload traces.

The results of this research show that spatial layout has a much greater effect on network contention than previously believed. We show that communication-sensitive and I/O-sensitive processor allocation algorithms can be designed which minimize the degree of network contention, and thus significantly improve overall performance. The results of this work has the potential to influence policies and decisions made within all dimensions of a high-performance system: from application program design to architectural configuration decisions to operating system libraries and resource management policies.

Our contributions are the following.

1. We analyze the impact of spatial layout of jobs on *inter-job link contention* due to communication between compute nodes under non-contiguous allocation. Results motivate the need for allocation compactness in order to minimize communication-based network contention.

2. We design a new *communication-sensitive* allocation algorithm that maximizes allocation compactness and thus minimizes inter-job link contention and improves the average response times of communication-intensive jobs, yet achieves all the advantages of non-contiguity.

3. We analyze the impact of spatial layout of jobs on *traffic hotspots* due to data transfer between compute nodes and I/O nodes. Results motivate the need for careful spatial layout of compute nodes in order to minimize parallel I/O-based network contention.

4. We design new *parallel I/O-sensitive* allocation algorithms that optimize the spatial layout of compute nodes relative to I/O nodes and thus alleviate traffic hotspots and improve the response times of I/O-intensive jobs.

5. Finally, we design new allocation algorithms that consider both sensitivity to parallel I/O and sensitivity to communication, and thus improve the performance under workloads that contain parallel I/O- and communication-intensive jobs.

The remainder of this dissertation is organized as follows: Background and related work is provided in Chapter II. Chapter III describes our problem model and research methodology. Chapters IV and V focus on communication-based network contention; Chapter IV analyses intra-job link contention and Chapter V presents a new communication-sensitive algorithm. Chapters VI and VII focus on parallel I/O-based network contention; Chapter VI analyses traffic hotspots and Chapter VII presents new parallel I/O-sensitive algorithms, as well as strategies that accommodate workloads that are both communication intensive and I/O intensive. Chapter VIII concludes the work.

CHAPTER II

BACKGROUND AND RELATED WORK

This chapter provides background information on the data transfer needs of high-performance applications and on the architecture of high-performance parallel computers. A detailed understanding of interconnection networks, of routing and switching, and of the I/O subsystem is needed to understand network contention and its effect on performance. This chapter also summarizes the state of the art in scheduling and allocation as a foundation for our approach to minimize network contention through the tuning of allocation strategies.

Architecture

A distributed-memory parallel system consists of a collection of processor nodes, each with local memory, that are connected to each other via an interconnection network. The means of data transfer between nodes is *message-passing* over the interconnection network.

Two classes of distributed-memory parallel systems are clusters and highly parallel systems. *Clusters* (also known as networks of workstations) typically consist of a few tens of workstations connected by a local area network such as Ethernet, FDDI or Myrinet. While each node of a cluster runs its own operating system, add-on software provides facilities for running parallel applications and managing the system as a single computer.

We concentrate on the second class of distributed-memory systems known as

highly parallel systems. They typically consist of hundreds or thousands of nodes. The operating system is more integrated, and specialized network hardware can achieve much higher bandwidths. Examples of highly parallel systems include the Intel TFLOPS (ASCI/ Red), the IBM SP2 and the Cray T3E.

Figure 2 shows the typical architecture of a node in a highly parallel system. A node consists of a processing element and a router. The processing element consists of one or more processors, local memory and an interface to the router. The router supports data transfer between nodes (also called interprocessor communication). Depending on the topology of the interconnection network, the router has several bi-directional channels, one channel to the local processing element and the rest to routers of other nodes.

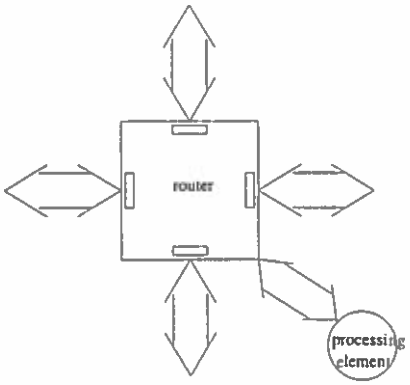


FIGURE 2. Architecture of a Node

For example, a node of the Intel TFLOPS consists of four off-the-shelf Pentium Pro processors, a node of the Intel Paragon consists of two i860 processors, and a node of the Cray T3E consists of one DEC Alpha processor.

Interconnection Network

Engineering and scaling reasons preclude large machines from having fully connected interconnection networks. We focus on *direct networks* where each node is directly connected to some neighbors in a point-to-point fashion. If sender and receiver are not directly connected, their message-passing requires multiple hops using multiple links.

Popular topologies of the interconnection network are hypercubes, meshes and tori. *Hypercubes* have low diameter and short path lengths which is very important if store-and-forward switching is used. The topology of the interconnection network of the Intel iPSC/860 is a hypercube. Figure 3a shows a four-dimensional hypercube.

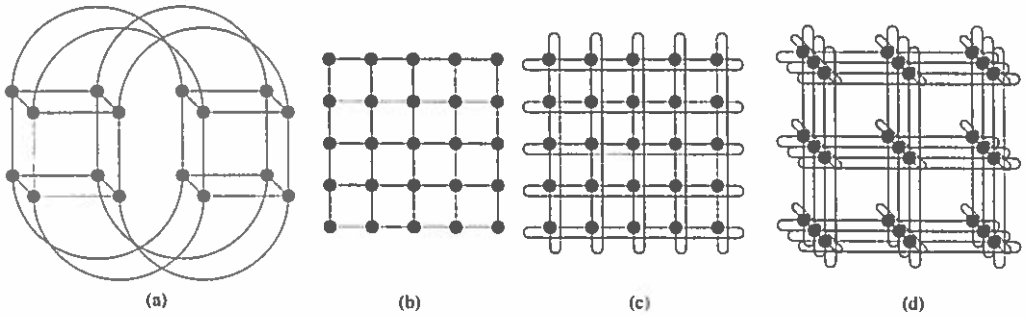


FIGURE 3. Four Examples of Topologies of Interconnection Networks

We concentrate on *mesh* topologies. In a mesh topology, each node is connected to both direct neighbors in each dimension. Meshes differ from hypercubes in that they are much simpler and easier to wire and that they can have wider channels and faster communication rates. Since wormhole switching (described later) reduces the importance of path length, the higher diameter of meshes is not a problem. A mesh topology is used in the Intel TFLOPS (ASCI/Red), the Intel

Paragon, the MIT Alwife and the Symult 2010 (previously Ametek). Figure 3b shows a two-dimensional mesh.

Adding wrap-around links between the first and the last nodes in each dimension transforms a mesh into a *torus.* A torus topology is used in the Cray T3E, Cray T3D, and a modified torus is used in the Tera MTA. Figure 3c shows a two-dimensional torus and Figure 3d shows a three-dimensional torus.

Hypercubes and tori belong to a class known as k-ary n-cubes. Some of our results for the mesh-interconnect are extensible to k-ary n-cubes as well.

Routing and Switching

Message-passing performance and the degree of network contention are highly dependent on the underlying routing and switching mechanisms. We concentrate on the current technologies of minimal dimension-ordered routing and wormhole switching. In *wormhole* switching [44], the flits of a message (the smallest units of data transmission) traverse the network in a pipelined fashion from the source to the destination node (see Figure 4). If the header flit is routed to a busy channel, the header and its trailing flits stop moving and block whichever channels they occupy in the network.

The message-passing delay of a packet transmitted through the interconnection network is composed of transmission delay and blocking time. *Transmission delay* is the interval between the time when the header flit of a packet enters the network at the source node and the time the tail flit arrives at the destination node, provided that there is no other message in the network (i.e. no network contention). Transmission delay determines a lower bound on the message passing

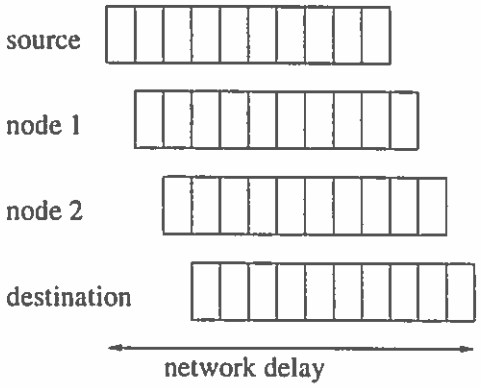


FIGURE 4. Pipelined Flow of Message Flits in Wormhole Switching

delay and is dependent on the switching technology used. For wormhole switching, the transmission time is $k_0 + k_1 * d + k_2 * (l - 1)$, where d is path length, l is packet length (number of flits) and k_0, k_1 and k_2 are system dependent constants. The first term, k_0, represents a fixed overhead. The second term, $k_1 * d$, represents the time for the header flit to set up a path, and the last term, $k_2 * (l - 1)$ is the time to transmit the remainder of the packet after the path is set up. Thus, transmission delay depends on path length and packet length. However, since the packet length is typically much larger than the path length, transmission delay depends primarily on the length of the packet, not on the distance it travels. Wormhole networks have the property that message-passing delay is almost independent of path length, provided that there is no other message in the network.

Transmission delay reflects the latency of packets sent in the absence of other network traffic. In contrast, *blocking time* reflects the dynamic behavior of the network resulting from the passing of multiple packets. Blocking occurs when several packets require the same network link. This is called network contention; we describe it in greater detail later in this chapter. Blocking time is high if the network

traffic is high or unevenly distributed. Our goal is to optimize resource management strategies such that blocking time due to network contention is minimized.

Routing determines the path taken for a packet traveling from a sender node to a receiver node. In *minimal dimension-ordered routing*, each packet is routed in one dimension at a time, arriving at the proper coordinate in each dimension before proceeding to the next dimension. In a two-dimensional mesh, the architecture we focus on, each node is represented by its position (x,y) in the mesh, and the minimal dimension-ordered routing algorithm is called XY routing. A packet to node (x_d,y_d) is sent first along the X dimension until it reaches the coordinate x_d and then along the Y dimension. Figure 5 shows an example. Minimal dimension-ordered routing avoids deadlock, allows no detours and is deterministic, i.e. there is only one path between a given source and a given destination. This architectural feature turns out to be important in our analysis of contention.

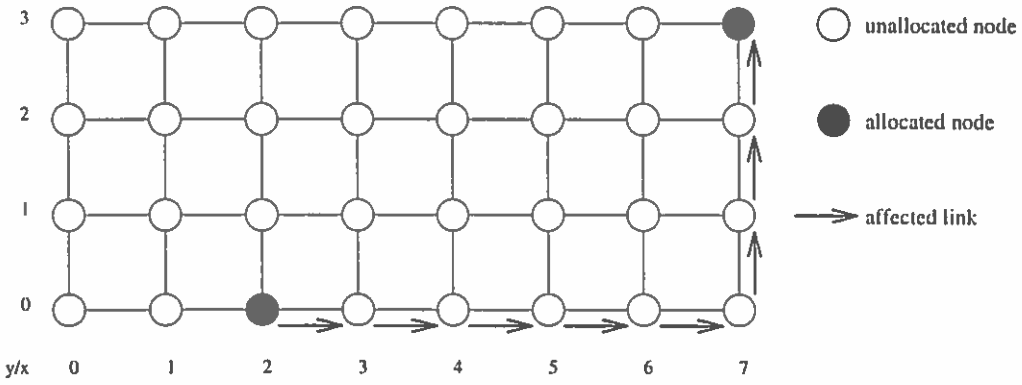


FIGURE 5. Minimal Dimension-Ordered Routing from (2,0) to (7,3)

I/O Subsystem

In order to achieve high-bandwidth, low-latency data transfer between memory and disks, the parallel I/O subsystems typically consists of a collection of *I/O nodes* [20], each managing and providing access to a set of disks (see Figure 6). Disks are typically arranged in RAIDs (redundant arrays of inexpensive disks), supporting different levels of fault tolerance.

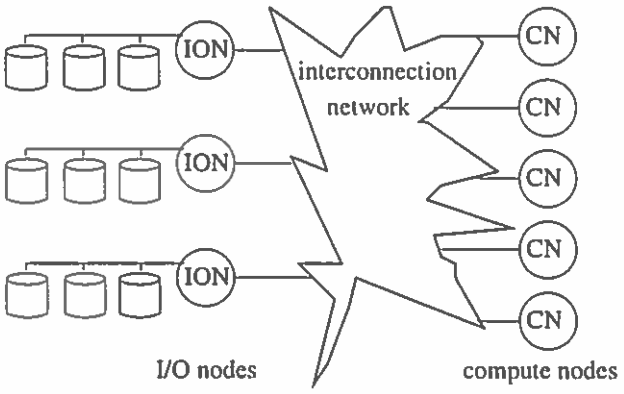


FIGURE 6. Typical Architecture of High-Performance Computers

In a parallel I/O system, individual files are striped across I/O nodes for performance. This improves the performance of a single access by allowing multiple disks to operate in parallel (see Figure 7). The distribution of file data among I/O nodes and disks is called the *file layout*. A file is broken down into file chunks, each assigned to one I/O node. As an example, 36 out of $32 \times 38 \times 2$ nodes on the Intel TFLOPS are I/O nodes. Each I/O node has 20 disks attached via two buses. Realistically, an I/O node bandwidth of $\approx$ 64 MB/s can be sustained. Each channel of the interconnection network can sustain a bandwidth of $\approx$ 380 MB/s per direction.

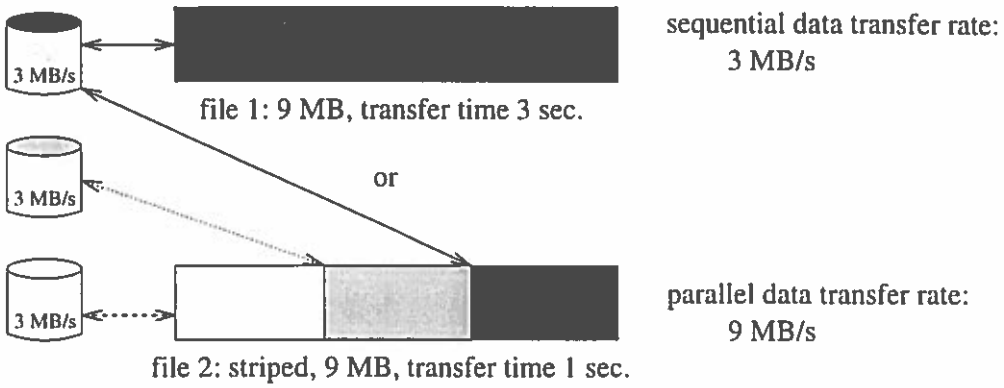


FIGURE 7. Example of Sequential Data Transfer of File 1 vs. Parallel Data Transfer of File 2

High-Performance Applications

High-performance applications can be classified as compute intensive, communication intensive, I/O intensive or a mix [50]. In all but compute intensive applications, the time incurred for data transfer is usually the most significant source of parallel processing overhead and can affect the job execution time significantly [31].

Communication

In order to take full advantage of available parallelism, parallel jobs distribute their data among compute nodes (see Figure 8). A variety of *data distributions* are supported by parallel compilers such as [27]. As a result, compute nodes have to communicate, and these pattern are often characterized by spatial and temporal regularities.

A classic example is the well-known n-body algorithm. Another example is an image-smoothing algorithm that performs a smoothing operation for each pixel of an image. Each smoothed pixel is the average value of the pixels in a window

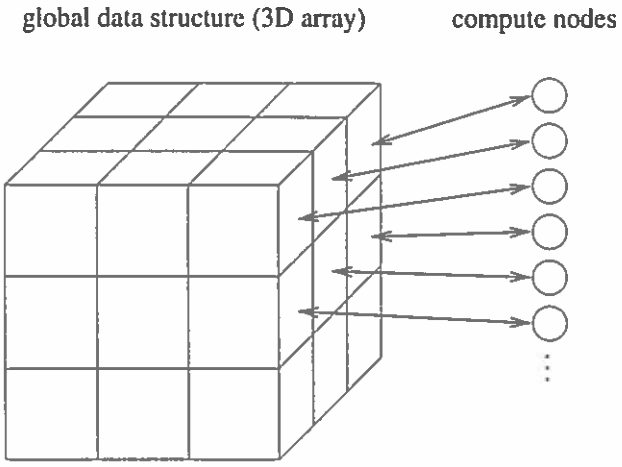


FIGURE 8. Global Data Structures Are Distributed Among Compute Nodes for Improved Performance.

centered around that pixel. Several pixels are distributed to each compute node. While each compute node can execute the smoothing operations for those pixels within the interior of its subimage, pixels along the border of each subimage have to be transferred between compute nodes. Additionally, synchronization requires message transfer between nodes.

There are a few basic communication patterns that are frequently used as building blocks in a variety of parallel algorithms.

1. In *one-to-all broadcast*, a single compute node sends identical data to all other compute nodes of the same job. The opposite pattern is single-node accumulation. Both patterns are used in several important parallel algorithms including matrix-vector multiplication, Gaussian elimination, shortest path, and vector inner product.

2. In *all-to-all broadcast*, each compute node of a job sends data to all other compute nodes of the same job. All-to-all broadcast is used in matrix operations, including matrix multiplication and matrix-vector multiplication. The

opposite pattern is multinode accumulation. The all-to-all broadcast pattern can also be used to perform other operations, such as reduction, prefix sum or scan.

3. In *one-to-all personalized communication*, a single compute node sends unique data to all other compute nodes of the same job. This operation is also known as single-node scatter. The opposite pattern is single-node gather.

4. In *all-to-all personalized communication*, also known as complete exchange, each compute node sends distinct data to all other compute nodes of the same job. Unlike all-to-all broadcast, each sender sends different data to different receivers. This operation is used in parallel fast Fourier transform (FFT), matrix transpose, and some parallel database join operations.

5. In *circular shift*, each compute node sends data to exactly one other compute node of the same job. This pattern finds application in some matrix computations, as well as in string and image pattern matching.

A large body of research [31, 6, 16] discusses efficient implementation of these basic communication patterns on various parallel architectures using store-and-forward or wormhole routing schemes.

Parallel I/O

Several studies analyzed the I/O requirements of typical scientific applications [39, 29, 49, 47, 13, 3]. I/O-intensive applications can be grouped into three basic categories: read-mostly, write-mostly and read-write. *Read-mostly* applications, such as data mining, read volumes of data, process the data, and typically

produce only a small amount of output. *Write-mostly* applications, such as simulations with visualization back-ends, begin with a small set of initial data, compute a step, write out the data, and repeat. *Read-write* applications, such as out-of-core solvers, repeatedly scan part of the data which is too big to fit into main memory, process that data, write it out, and repeat.

Scalar Pentadiagonal (SP) and Block Tridiagonal (BT), part of the NAS Parallel Benchmarks [4], are examples of I/O-bound applications. These two applications account for a large portion of the computational fluid dynamics workload at NASA Ames [46]. In both applications, a three-dimensional grid of variables is repeatably computed and saved to a striped file which can be processed by visualization software. A parallel implementation of SP and BT that provides good load balancing and coarse-grain communication employs the Multi-partition method [46]. This method uses the following *data distribution*: If there are n compute nodes, the three-dimensional grid of data items is broken up into $n\sqrt{n}$ sub-cubes. The location of the $\sqrt{n}$ sub-cubes per compute node is chosen such that for any plane of sub-cubes, each compute node gets exactly one sub-cube from that plane (see Figure 9a).

One possible *file layout* is shown in Figure 9b. A file represents a linearization of the global data structure, such as the row-major ordering of a three-dimensional array. This linearization is often called *canonical file*. In a striped file, chunks of the canonical file are distributed across I/O nodes, typically in a round-robin fashion. It is easy to see that for the data distribution of the Multi-partition method, every compute node has to write to every I/O node, regardless of the stripe size.

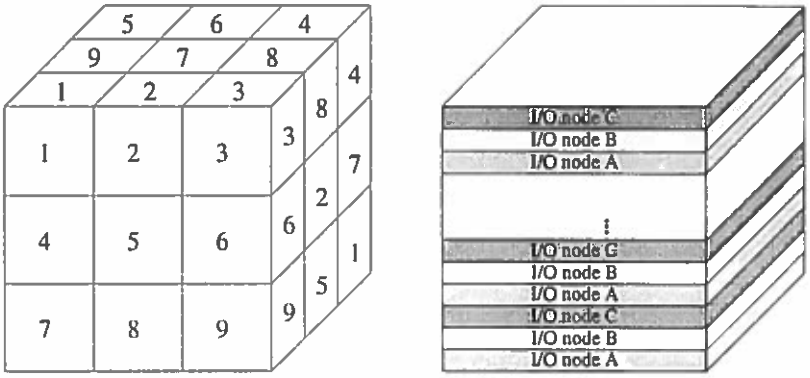


FIGURE 9. Data Distribution and File Layout of a Three-Dimensional Grid of Data Items for a Scientific Application

Scheduling and Allocation

A high-performance parallel computer is an expensive resource that is typically *shared* among a community of users. As part of the operating system, the system scheduler is in charge of *parallel job scheduling*, i.e. allocating resources to competing jobs. The goal is to minimize the response time of jobs and maximize the system throughput over a stream of jobs. Parallel job scheduling is not to be confused with the scheduling of tasks within a parallel job which is done by the programmer or the runtime system.

Given a workload of jobs, the system scheduler has to decide when to run each job and how many and which compute nodes to assign to each job. Most high-performance parallel systems employ space sharing and variable partitioning. In *space sharing*, multiple jobs run on disjoint subsets of the processor nodes. *Variable partitioning* [19] means that each job is assigned as many compute nodes as it requested and that each job runs on those nodes for its entire lifetime.

Under the assumptions of space sharing and variable partitioning, the sched-

uler decides (1) in what order to run the jobs and (2) what *spatial layout* of nodes to assign to each job (see Figure 10). The former decision is made by the *job scheduling strategy.* This strategy controls the temporal admission and queuing of jobs. Previously proposed strategies include first-come first-served (FCFS), smallest job first (SJF), backfilling [51] and scan [30]. These policies are used in commercial scheduling systems including NQS, LoadLeveller, EASY, PBS and PSched. Our focus is on the latter decision, made by the *processor allocation strategy.*

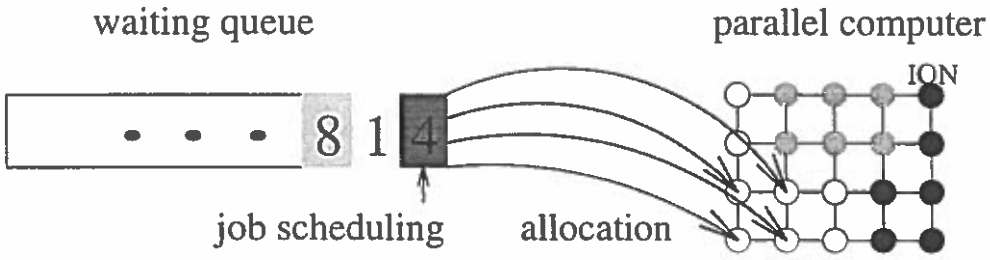


FIGURE 10. Job Scheduling and Processor Allocation

Contiguous Processor Allocation Strategies

Processor allocation (also known as system partitioning) is the problem of assigning a portion of the processor "space" to each scheduled job. Early processor allocation algorithms allocated nodes in a "contiguous" manner only. Intuitively, contiguity means that all the nodes allocated to a job form a *convex shape* such that all message-passing between nodes of a job stay within the job's area. This constraint often is enforced by requiring the nodes allocated to a job to form a subgraph of the original topology (e.g. a submesh in a mesh). Figure 11 shows an example of four jobs allocated contiguously in a two-dimensional mesh topology.

Contiguity ensures that messages from different jobs do not interfere with

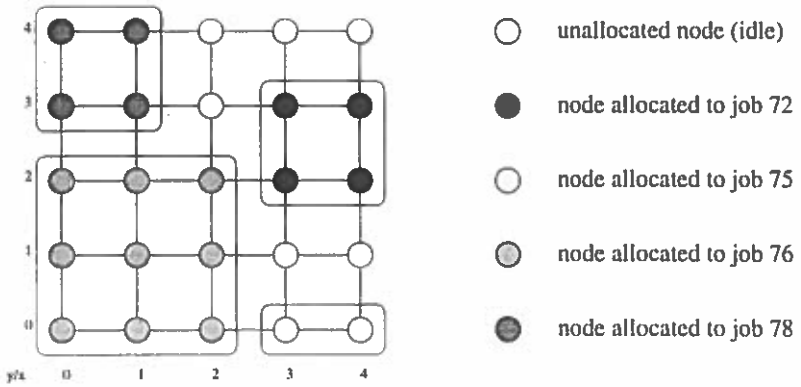


FIGURE 11. Snapshot Showing Contiguous Allocations of Four Jobs

each other even if the jobs run at the same time. Thus, each job can think of owning the machine exclusively, and several runs of the same job result in almost equal execution times.

Contiguous processor allocation was an area of research for about a decade until the mid 1990's. Examples of contiguous allocation strategies for mesh topologies are 2D Buddy [32], Frame Sliding [11] and First and Best Fit [58]. For hypercube topologies, examples are Gray Code [8] and Partners [1].

The key disadvantage of contiguous processor allocation strategies is fragmentation. Consider an additional job in Figure 11 that requests four nodes. Six nodes are unallocated at this time, but there is no contiguous block of four nodes available. The nodes (2,3), (2,4), (3,4) and (4,4) for example are not contiguous because they do not form a convex shape. This situation is called *external fragmentation.*

Additionally, *internal fragmentation* can arise from requesting j nodes but getting $i > j$ nodes because of constraints of the machine or constraints of the allocation strategy. Both types of fragmentation result in unused nodes and thus

low *utilization* of the machine. Typical utilizations of only 34% to 66% for contiguous allocation strategies are reported [32, 58, 30, 33]. This is unfavorable and contradicts the goal of high throughput over a stream of jobs.

To solve the fragmentation problem, *non-contiguous* processor allocation strategies have been proposed [25, 33, 55, 53]. These strategies allocate nodes that are possibly dispersed throughout the system (see Figure 12). These strategies are also called scattered or fragmentation-free allocation strategies.

Non-Contiguous Allocation Strategies

Non-contiguous strategies experience neither internal nor external fragmentation and thus outperform contiguous strategies reaching utilizations of up to 78% for common workloads [33, 55, 53]. The bigger physical distance between dispersed nodes allocated to the same job turns out to not be a problem because – due to wormhole switching – the network latency is almost independent of the path length, as long as messages do not interfere. However, as we will see later, the potential impact from messages that do interfere cannot be ignored.

For mesh topologies, the following non-contiguous strategies have been proposed:

(1) "Paging" [33] scans the topology in fixed order for unallocated nodes, e.g. row by row in the mesh. Paging was shown to have very good performance [35] and was used at NASA Ames Research Center for their Intel Paragon.

(2) "Multiple Buddy Strategy" (MBS) [33] typically allocates several contiguous blocks to a job. It is a hierarchical strategy and maintains *free lists*, i.e list of blocks that consist of idle nodes of sizes $2^i \times 2^i$, $0 \leq i \leq \frac{logN}{2}$. Each request is bro-

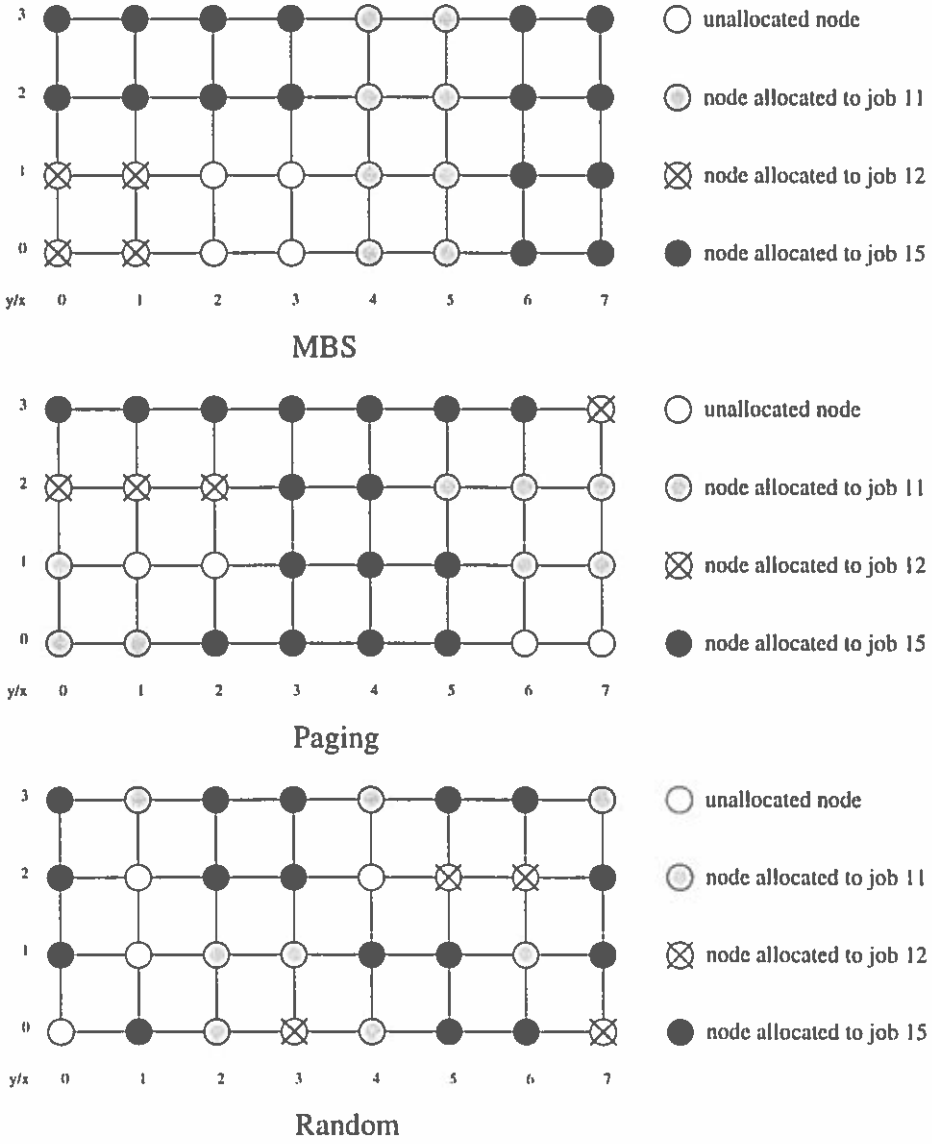


FIGURE 12. Snapshots Showing Different Non-Contiguous Strategies

ken down into base-four subrequests and MBS attempts to satisfy these requests with the first blocks in the corresponding free lists. Bigger blocks or subrequests are only broken down further if this is necessary to avoid external fragmentation. As an example, Figure 13 shows a snapshot where two blocks of size 2×2 and two blocks of size 1×1 were available. The next job requests $2 \times 3 = 6 = 4 + 1 + 1$ nodes, and thus one 2×2 and both 1×1 blocks are assigned.

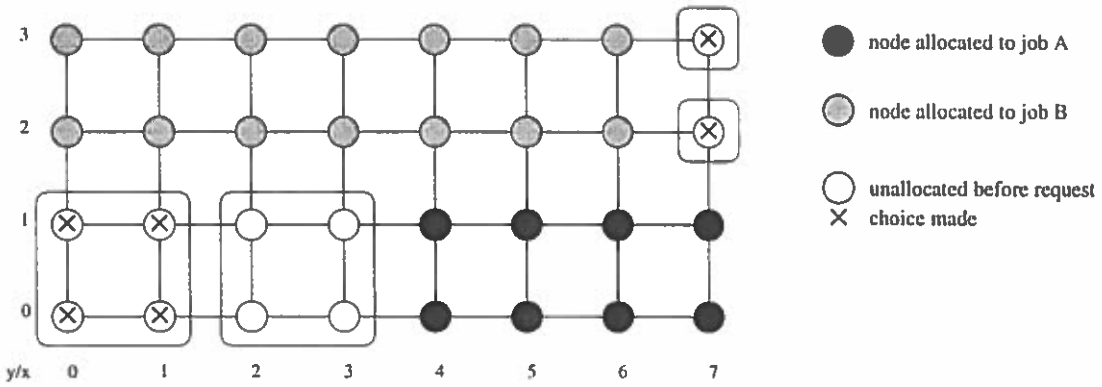


FIGURE 13. How MBS Allocates a Job of Size 2×3

(3) "Modified 2-dimensional Buddy" (M2DB) [53], in use on the SDSC Paragon, is similar to MBS. However, if there are choices, it satisfies subrequests with blocks from the corresponding free lists that are at minimal distance from the block allocated to the biggest subrequest.

(4) "AS&MB" [52] is a hybrid strategy employing the non-contiguous MBS strategy if the contiguous Adaptive Scan strategy [14] fails.

[55] describes similar algorithms for k-ary n-cube topologies.

Previous research [33, 53, 42] showed that non-contiguous processor allocation strategies are very successful in increasing system utilization by avoiding fragmentation. However, if jobs transfer data due to communication or I/O, the

increased potential for network contention due non-contiguous allocation can be a problem. Lo et al. [33] showed that non-contiguous allocation strategies perform better overall than contiguous ones – even when network contention is considered – and that non-contiguous allocation strategies that provide some degree of contiguity exhibit the best performance. To obtain optimal performance, processor allocation strategies have to consider network contention due to communication and I/O needs of applications.

Network Contention

Communication- and I/O-bound jobs transfer large amounts of data. Network contention occurs when multiple data transfers use the same communication link at the same time. Each link of the interconnection network can only be used by one packet at a time, other packets have to wait.

There are two kinds of network contention. *Intra-job link contention* or *internal contention* occurs when two or more data transfers from the *same* job contend for a link. In contrast, *inter-job link contention* or *external contention* occurs when two or more data transfers from *different* jobs contend for a link. Figure 14 shows an example of inter-job link contention. Data transfer from job A and data transfer from job B both use the link between node (6,0) and node (7,0).

Both kinds of network contention affect the data transfer time of messages, execution time of jobs, and ultimately job throughput of the machine. Although contention has been shown to be neglectable in case of small messages [33, 42] or high software latency [42], it in general is the deciding performance factor within the class of non-contiguous processor allocation strategies.

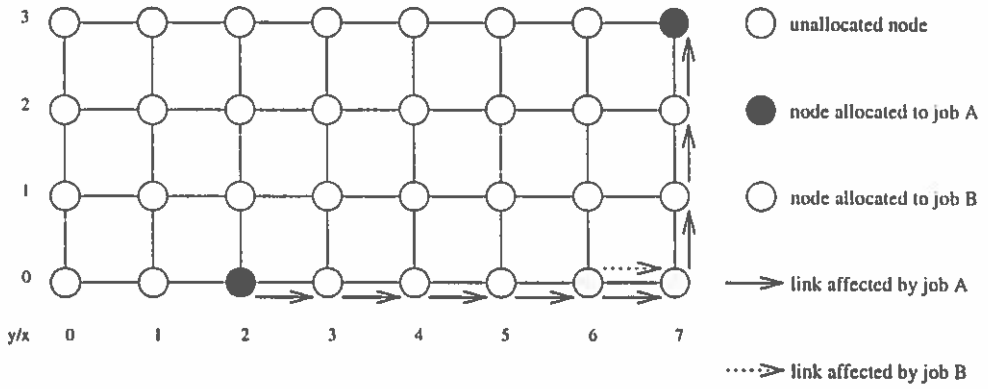


FIGURE 14. Inter-Job Link Contention for the Link Between (6,0) and (7,0)

The research on communication-based network contention involves both simulation-based and empirical studies on actual parallel machines. Regarding internal contention, Chittor and Enbody [9] investigated the communication performance degradation caused by network contention. They performed measurements on a Symult 2010, an Intel DELTA and an Intel Paragon. Their results showed that contention for network links increased the communication time per message exponentially as the communication-to-computation ratio increases. Bokhari et al. [6] investigated whether it is wise to trade-off contention cost with synchronization cost. Their measurements on an Intel Paragon showed that contention-bounded message schedules perform best. Eberhart and Li [16] devised contention-free message schedules for shift and transpose patterns. Their simulations showed improvements over random schedules. Moritz et al. [43] introduced contention into the LogP model.

Regarding inter-job link contention, Min and Mutka [40] developed a contention model. Nitzberg, Lo et al. [33] performed contention measurement on an Intel Paragon. They showed that for large messages, contention grows lin-

early as a function of the number of competing messages on a link. Based on their observations, they concluded that non-contiguous allocation strategies must take contention into account. Moore and Ni's [42] results indicated that in systems with low software latency, allocations with less contention result in faster execution times for communication-intensive applications.

Research on network contention due to I/O traffic is limited. Baylor et al. [5] investigated the I/O performance of the IBM Vulcan, the predecessor of the IBM SP series. Their results showed substantial degradation in I/O performance due to network saturation as the I/O request rate increases. Winslett et al. [10] optimized the placement of half-time I/O nodes in a network of workstations in order to avoid data transfers and thus minimize network contention. Garg et al. [23] performed I/O measurements on an Intel TFLOPS machine with up to nine I/O nodes and several hundred processor nodes. Their results showed that certain placements of compute nodes and I/O nodes saturated the bandwidth of some network links, resulting in serious parallel I/O performance degradation. Feitelson et al. [20] and Kotz et al. [28] mention network contention as a concern but do not investigate further.

In summary, no one has conducted a systematic analysis of the effects that the processor allocation strategy, through its spatial layout decisions, has on network contention, and little effort has been made to tune processor allocation strategies for communication and I/O-intensive workloads.

CHAPTER III

PROBLEM DESCRIPTION AND APPROACH TAKEN

Minimizing Network Contention

Any non-trivial application has to transfer data due to communication or I/O needs. The high cost of data transfer over the interconnection network, which is a serious bottleneck to high-performance parallel computing, is exacerbated by network contention. Therefore, it is crucial to minimize network contention in order to achieve fast job response times and high system throughput.

Network contention can be reduced through several strategies which fall into three main categories: elimination of data transfers, scheduling of data transfers so that they do not contend in time or spatial rearrangement of data transfers. These different strategies are illustrated using the simple example of Figure 15 showing the original situation where one node on the right receives data from three nodes on the left. In this figure, the underlying mesh topology is not shown. Contention occurs because messages sent from the top two nodes contend for a common link on their way to the target node.

(a) *Eliminating* data transfers by modifying the traffic pattern. If the node on the right only receives data from the lower two nodes on the left, as shown in Figure 15a, one data transfer from the original situation is avoided and no contention occurs. A similar approach has been proposed by Winslett et al. [10]. By placing half-time I/O nodes in network of workstations so

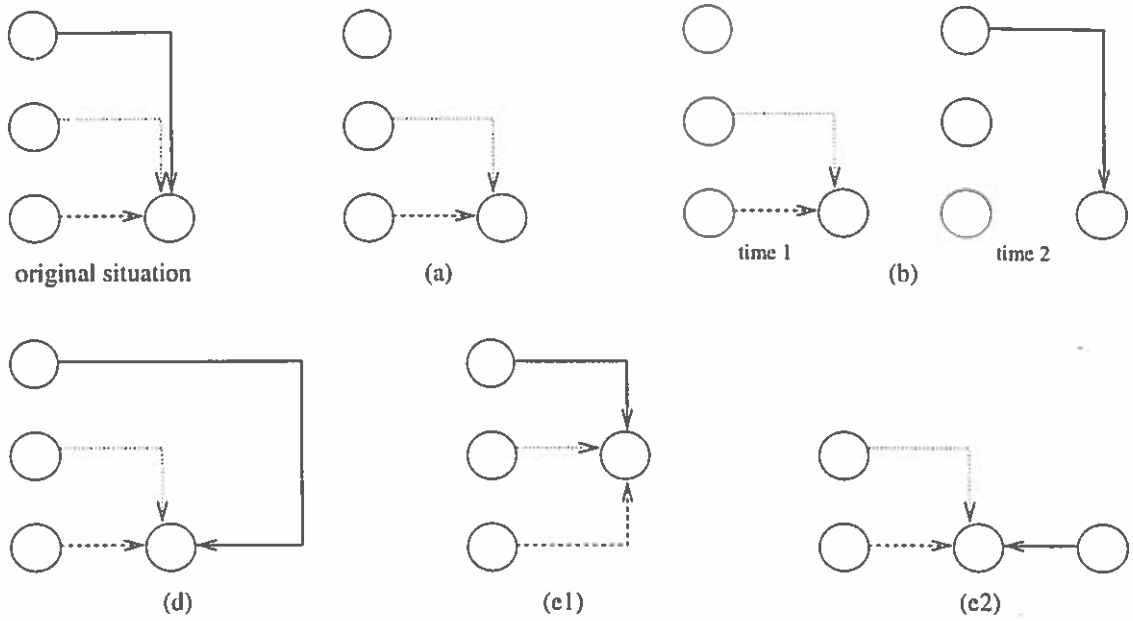


FIGURE 15. Strategies for Minimizing Network Contention: Original Situation (Upper Left), Eliminating Data Transfers (a), Scheduling Data Transfers at Different Times (b), Modifying Routing (d), Modifying the Spatial Layout of Nodes (e1, e2)

that local data can be exploited, they eliminated data transfers. However, in general situations, modifying the traffic pattern may not be an option. The traffic pattern depends on the parallel algorithm, the data distribution across compute nodes and the file layout across I/O nodes, all of which are determined by the application, the compiler or the system configuration.

(b) *Scheduling* different data transfers at different time slots can reduce the number of data transfers using the same network link at the same time. In Figure 15b, the data from the lower two nodes on the left is sent at time 1, while the data of the upper node on the left is sent at time 2. As a result, no contention occurs. Algorithms and heuristics to find contention-free message schedules have been proposed by Jain et al. [26], by Eberhart and Li [16]

and by Bokhari et al. [6]. However, possible problems include the high synchronization cost that has to be paid, the possible deterioration of disk seek times due to reordered data transfers, as well as the fact that the architecture and system software of most parallel computers currently do not permit sufficiently fine-grained control over the timing and order of I/O operations [26].

(c) A more coarse-grain approach is to modify the *temporal job mix* by limiting the number of communication- and/or I/O-bound jobs scheduled to run concurrently. We address this option in Chapter VIII.

(d) Modifying *routing* can result in different network links taken. In the example of Figure 15, if the data from the node in the upper left deviates from minimal XY routing and takes the path shown in Figure 15d, no contention occurs. It has been proposed to provide additional links to the interconnection network [54] or to use adaptive routing [44]. However, for reasons of low overhead and to avoid deadlock, routing in high-performance parallel computers is most often fixed to minimal dimension-ordered routing.

(e) Modifying the *spatial layout* of nodes can result in given data transfers using different network links, even if routing is not changed. In Figure 15e1, the location of the target node is changed. In Figure 15e2, the node formerly in the upper left changed location. Both changes in spatial layouts result in no contention occurring.

In this work, we concentrate on strategy (e), on tuning the spatial layout of the compute nodes in relation to the I/O nodes which are at fixed location within

the interconnection network topology.

Our Approach: Tuning the Spatial Layout of Jobs

Our approach is to allocate processors to jobs in a more careful – parallel I/O- and communication-sensitive – way. Changing the spatial layout of jobs can have a significant effect on network contention. An example of how spatial layout impacts *communication-based* network contention is shown in Figure 16. Given the situation in Figure 16a and a new job D requesting eight nodes, the allocation strategy has to choose among 12 idle nodes. Two possible spatial layouts are shown in Figures 16b and 16c. There is a significant difference in inter-job link contention if all four jobs do all-to-all personalized communication. In the layout shown in Figure 16b, communication of job D interferes with communication of all other jobs. In contrast, in the layout shown in Figure 16c, communication of the job D interferes only with communication of job A.

An example of how spatial layout impacts *I/O-based* network contention is shown in Figure 17. The allocation strategy has a choice between 4 idle nodes to assign to a job of size two. Two possible spatial layouts are shown. There is a significant difference in internal contention if the job does parallel I/O and writes data from each compute node to each I/O node. For the layout shown in Figure 17a, four data transfers are contending for the link left of CN1. In contrast, in the second layout, the maximum number of data transfers contending for a link is two.

We consider three aspects of spatial layout: compactness, shape and location.

1. The spatial layout of job A is more "compact" if communication among

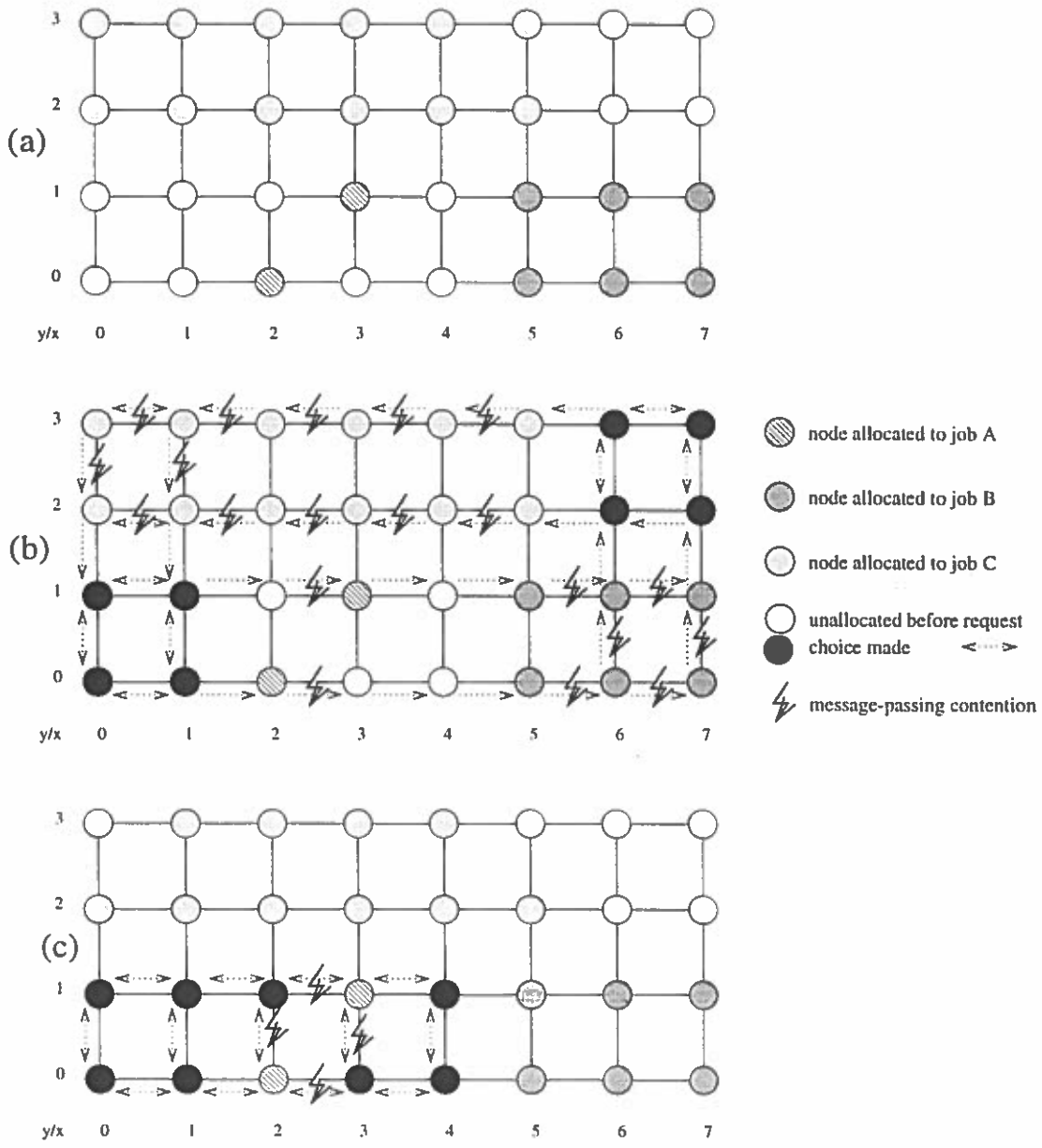


FIGURE 16. Modifying Processor Allocation Has an Effect on Communication-Based Network Contention

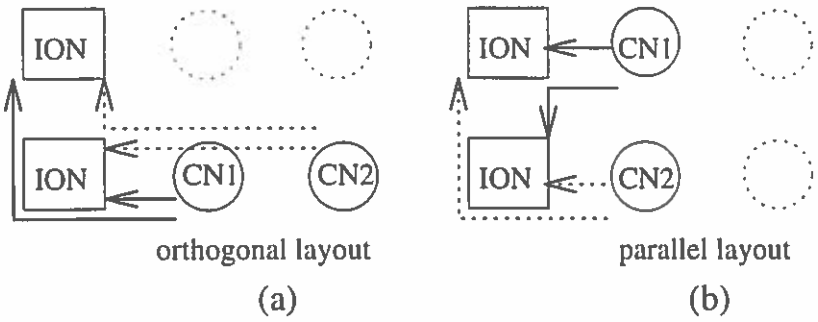


FIGURE 17. Modifying Processor Allocation Has an Effect on I/O-Based Network Contention

nodes of job A affects fewer foreign nodes (nodes not assigned to job A). In the communication example of Figure 16, the spatial layout of job D in Figure 16c was more "compact" than the spatial layout of job D in Figure 16b. Rectangular contiguous layouts are as compact as possible while layouts that occupy only the first and last column of the mesh are the worst, "sandwiching" all the compute nodes in the whole machine. In situations where it is impossible to allocate the job contiguously (e.g. the situation shown in Figure 16), possible layouts differ in degree of compactness. In Chapter IV, we define *dispersal metrics* that capture the opposite of compactness: dispersal. In Chapter V, we aim at separating jobs such that their communication traffic interferes as little as possible.

2. *Shape* is a characteristic of contiguous allocations. In the I/O example of Figure 17, the job in Figure 17a has a horizontal shape which is orthogonal to the I/O nodes, whereas in Figure 17b, the job has a vertical shape which is parallel to the I/O nodes. For non-contiguous allocations, the shape of a job is defined by the shape of the rectangle enclosing all its nodes. The approaches used by current commercial allocation software fall into two categories: block-

based allocation schemes [53] or linear allocation [41, 45]. In Chapter VI, we consider the basic shapes: block, vertical, horizontal and diagonal. While it is in general not possible to separate the I/O traffic of one job from the I/O traffic of another job, we aim for even distribution of I/O traffic such that hotspots are alleviated (Chapter VII).

3. Besides compactness and shape, there often remain choices of different *locations*. These include distance from other jobs, closeness to I/O nodes or positions in the middle of columns (which can result in even traffic distributions if I/O nodes are placed horizontal, see Chapter VII).

Our goal is to design parallel I/O- and communication-intensive allocation strategies that minimize network contention. We work towards this goal in the following manner. First, we focus on communication-based network contention. We investigate the effect of allocation compactness on network contention. Applying our results, we design new allocation algorithms that are communication-sensitive. Then, we focus on I/O-based network contention. We investigate the effect of spatial layout (shape and location) on network contention. Applying our results, we design new allocation algorithms that are parallel I/O-sensitive. Finally, we design new allocation algorithms that consider both the sometimes conflicting goals of sensitivity to parallel I/O and sensitivity to communication, and thus improve the performance under workloads that contain parallel I/O- and communication-intensive jobs.

For both communication traffic and parallel I/O traffic, we develop a model and appropriate metrics for measuring contention. Through graph theoretic analysis and dynamic simulation, we analyze the relationship between spatial layout,

network contention, and throughput. We evaluate the performance of our new allocation algorithms through simulation using both stochastic and real workload traces.

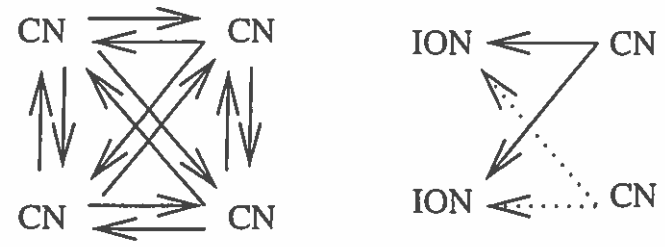
Analytic Modeling

To analyze the effects of spatial layout on network contention, we conduct a careful static analysis using a graph theoretic model and looking very closely at communication and I/O intensive jobs.

Communication Traffic Model

We concentrate on the communication pattern of all-to-all personalized broadcast, also called "complete exchange". This pattern is at the heart of parallel algorithms for matrix transposition, matrix-vector multiplication, the fast Fourier transform and a method for solving partial differential equations. This pattern requires data movement from each compute node to all other compute nodes, and can be represented as a complete directed graph (see Figure 18a). It is the densest possible communication requirement, and the time spent executing it is an important performance parameter.

We also consider one-to-all broadcast, n-body computation, fast Fourier transform (FFT) as well as those patterns executed by the multigrid benchmark and the kernel conjugate gradient (CG) benchmark from the NAS parallel benchmarks [4].



communication traffic graph (a)

I/O traffic graph (for write) (b)

FIGURE 18. Communication and I/O Traffic Patterns

I/O Traffic Model

We assume an I/O traffic pattern in which every compute node exchanges data with every I/O node and data transfered in each message is unique. This I/O traffic pattern, which can be modeled by a complete bipartite graph (see Figure 18b), is representative of a wide range of scientific applications in which a large data structure is read from or written to a file striped across I/O nodes and is then re-distributed to the compute nodes according to one of the classic data distribution patterns, such as those defined in High Performance Fortran (HPF) [27].

Graph Theoretic Model

We define *communication traffic graphs* and *I/O traffic graphs* to model the data transfer needs of jobs. A *system graph* defines the target architecture.

Definition: The *communication traffic graph* of a job is a directed graph $G_{comm} = (V_{comm}, E_{comm})$ where each vertex represents a task of the parallel job, $| V_{comm} |$ equals the number of compute nodes required by the job, and each edge

$(V_i, V_j) \in E_{comm}$ indicates data transfer from V_i to V_j.

We concentrate on the all-to-all personalized communication pattern; its communication traffic graph is a complete graph.

Definition: The *I/O traffic graph* of a job is a directed bipartite graph $G_{I/O} = (I_{I/O}, C_{I/O}, E_{I/O})$ where $I_{I/O}$ represents I/O nodes, $C_{I/O}$ represents the tasks of the parallel job, and each edge of $E_{I/O}$ indicates data transfer for I/O.

In a read traffic graph, all edges go from $I_{I/O}$ to $C_{I/O}$. In a write traffic graph, all edges go from $C_{I/O}$ to $I_{I/O}$. We concentrate on the I/O traffic pattern in which every compute node exchanges data with every I/O node, whose I/O traffic graph is a complete bipartite graph.

Definition: The *system graph* of a machine is a directed graph $G_{sys} = (V_{sys}, E_{sys})$, $V_{sys} = V_{ION} \cup V_{CN}$, where V_{ION} represents I/O nodes and V_{CN} represents compute nodes and E_{sys} represents channels corresponding to the topology of the interconnection network.

We concentrate on mesh topologies. We assume a two-dimensional numbering scheme for the nodes where $N_{0,0}$ is the node in the lower left hand corner and $N_{a,b}$ is a to the right and b up. An edge between two nodes $N_{i,j}, N_{k,l} \in V_{sys}$ is in E_{sys} if either $i = k$ and $\mid j - l \mid = 1$ or $j = l$ and $\mid i - k \mid = 1$. We concentrate on machine configurations where the I/O nodes occupy the leftmost column of the mesh, i.e. $V_{ION} = \{N_{0,0}, N_{1,0} \ldots N_{k,0}\}$.

Definition: An *allocation a* assigns compute nodes to a job with communication traffic graph G_{comm} such that $a(V_{comm}) = S \subseteq V_{CN}$ and $\mid S \mid = \mid V_{comm} \mid$. Similarly, an *allocation a* assigns nodes to a job with I/O traffic graph $G_{I/O}$ such that $a(C_{I/O}) = S \subseteq V_{CN}$, $\mid S \mid = \mid C_{I/O} \mid$, with the additional constraint $a(I_{I/O}) = V_{ION}$.

Intuitively, the allocation of a job doing I/O can be modeled by *embedding* the I/O traffic graph into the system graph subject to the following constraints:

1. the m I/O nodes of the I/O traffic graph are mapped bijectively to the m I/O processors in the system graph;

2. the n compute nodes of the I/O traffic graph are mapped injectively to some subset of the compute processors in the mesh;

Due to multiprogramming, there can be several jobs in the system at the same time. At any instant t, for two jobs $i \neq j$ running at the same time, their allocated compute nodes do not overlap, i.e. $a(V^i) \cap a(V^j) = \emptyset$. However, the I/O nodes are shared among all jobs, i.e. $a(I^i) = a(I^j) = V_{ION}$.

Definition: The *path* $a(e)$ of a data transfer is the image of an edge $e = (V_i, V_j)$ of the job's communication traffic graph (or I/O traffic graph), given an allocation a. The path is a sequence of edges of the system graph defined by the routing function for data transfer between nodes $a(V_i)$ and $a(V_j)$.

For dimension-ordered XY-routing, if the endpoints of an edge in G_{comm} (or $G_{I/O}$) are mapped to $N_{k1,l1}$ and $N_{k2,l2}$ for the case where $k1 \leq k2$ and $l1 \leq l2$, the path includes all vertical edges of the system graph between nodes $N_{i,l1}$ and $N_{i+1,l1}$ for $k1 \leq i < k2$ as well as all horizontal edges between nodes $N_{k2,j}$ and $N_{k2,j+1}$ for $l1 \leq j < l2$. Similar constraints hold in the case where $k1 > k2$ or $l1 > l2$.

Intuitively, each edge from I/O node i to compute node j in the I/O traffic graph is mapped to the path from I/O processor $a(i)$ to compute processor $a(j)$ according to the XY-routing scheme of the mesh.

Two important metrics for our analysis are link contention and max_contention.

Definition: *Link contention* on link l occurs if for two edges $e_i \neq e_j, l \in a(e_i) \wedge l \in a(e_j)$. It is intra-job link contention if e_i and e_j are members of the edge set of the same job. It is inter-job link contention if their are members of the edge set of different jobs.

Definition: A link with maximal link contention is a *hotspot* and the value of this maximum contention is denoted by $max\_contention$.

Simulations

The second phase of our research involves dynamic simulations in order to further investigate the relationship between spatial layout and network contention. Our analysis leads to the development of parallel I/O- and communication-sensitive allocation strategies. They are all non-contiguous algorithms that achieve the benefits of fragmentation-free approaches while minimizing network contention. Our new allocation strategies can be tested using simulations to model the dynamic behavior of mesh-based systems.

All experiments were conducted using ProcSimity [57], a simulation tool developed for the evaluation of job scheduling and processor allocation algorithms for distributed-memory parallel computers. ProcSimity models a variety of network topologies and several current flow control and routing technologies. ProcSimity supports a variety of communication and I/O traffic pattern. Message-passing can be simulated in detail at the flit level or at the packet level. ProcSimity's visualization and performance analysis tools allow for viewing a dynamic animation of the selected algorithms as well as a variety of system and job level performance metrics. ProcSimity has been successfully used in experiments investigating the

feasibility of non-contiguous processor allocation.

ProcSimity is built around a model in which independent user jobs arrive in the system, requesting a particular sized partition of the system's processors. If an arriving job can not be run immediately, due to a lack of free processors or other waiting jobs, the job is diverted to the system waiting queue. Before a waiting job can leave the waiting queue, the *scheduler* must place the job at the head of the queue, and the *allocator* must determine that a partition can be constructed for the job from free processors in the system. When a job is ready to be run, the allocator assigns the job the needed processors, depending on the allocation strategy used. Although messages from other jobs may pass through this new partition, the new job holds these processors exclusively until it finishes running. At this time, it departs the system and its processors are freed for use by other incoming jobs.

The ability to accurately model the arrival and servicing of jobs in the system is a very important factor affecting the quality and validity of the simulation results obtained. ProcSimity can be driven by probabilistic or real workload traces, and running jobs can be simulated at three different levels of detail. First, jobs can simply delay for a certain time and then depart. This method is best for experiments in which the service time for a job is assumed to be independent of other jobs and independent of its spatial layout in the architecture. This allows us to test and isolate the effects of different combinations of scheduling and allocation algorithms on system fragmentation alone, without simulating specific applications.

The second and third level of detail simulates communication and I/O patterns, with individual messages simulated either at the level of packets (second

level) or at the level of flits (third level) moving through the network. These methods allows for a more realistic comparison of different strategies by taking into account message passing overhead. This is important because message-passing delay may vary depending on the spatial layout of the allocation, and thus, on the amount of message contention between different jobs.

Performance information is provided to ProcSimity's visualization tool through trace files generated by its discrete event simulator, modeling a distributed memory multicomputer in a multiple-user environment. The simulator is implemented in C, using the process-oriented, discrete event simulation toolkits YACSIM, a general simulation library, and NETSIM, a library of network simulation extensions [12]. The visualization tool was built using Tcl version 7.3 and Tk version 3.6.

High level simulations such as ProcSimity pose a significant challenge in the validation of simulation results. Because the simulation model is simplified, showing only relative worst-case performance, simulation results cannot be compared directly with results from real parallel computers. However, we have made extensive efforts to verify and validate our simulation model and its implementation.

Output analysis, statistically studying the degree of variance in results obtained from duplicate simulation runs, gives an indication of the reliability of the simulation apart from any real system being modeled. All ProcSimity statistics can be measured with less than 4% error given 95% confidence and a sufficiently long simulation (at least 10 replicated runs with 1000 jobs each). Further, we validated our simulator by comparing its results to other independently developed simulations of similar systems. Windisch et al. [57] conducted experiments compar-

ing our simulator, without job communication, to Zhu's allocation simulator[58], which simulates jobs according to the same probabilistic service delay scheme used by ProcSimity. We also conducted experiments comparing our simulator, with job communication patterns, to the Warp simulator [15] and to the Chaos simulator[21], which we modified to execute independent jobs and ProcSimity's communication patterns. In all three comparisons, nearly identical normalized results were obtained, differing by no more than 7% in the worst case. Therefore, through extensive code verification, testing, output analysis, and comparison with other simulators, we conclude that ProcSimity provides an accurate and meaningful facility for comparing the effects of scheduling and allocation strategies on parallel computer performance.

Workloads

Simulation-based performance evaluation of resource management strategies can be driven by either *synthetic workload models* that use probability distributions or by *real workload traces* gathered from scientific production runs on real supercomputers. We use both methods. *Real workload traces* captured from production machines can provide a very high level of realism when used directly in performance evaluation experiments. A workload trace is a record of resource usage data about a stream of parallel jobs that were submitted to and run on a given message-passing parallel machine. For our experiments, we use a real workload trace from a 400 node Intel Paragon housed at the San Diego Supercomputer Center [56]. More details are reported later.

In contrast, *synthetic workload models* offer the convenience of a much more

manageable experimental medium that is free of the idiosyncratic site-specific behavior of production traces. This is both its advantage and a potential point of criticism – the use of a tractable synthetic model provides a neutral, more universal experimental environment but does not yield the realism and pragmatic testing desired by some researchers.

Our simulator models a stream of jobs that arrive, execute for a period of time, and then depart the system. The simulator is driven by a synthetic or real workload traces that includes job arrival time and jobsize. Modeling the work requirement of each job, we need to specify the internal job structure. As stated in [18], there is not much hard data that has been measured about typical distributions of internal job structures. However, it is clear that the most common and clearly identifiable structures are the computational structure (parallelism and barrier synchronizations), memory requirements, interprocess communication, and I/O needs. We focus on the latter two. Each job in our simulator sends messages according to the following traffic patterns: heavy communication traffic (all-to-all pattern), heavy parallel I/O traffic (complete bipartite), or both communication and parallel I/O traffic.

Performance Metrics

We use the following performance metrics:

1. *average response time*: the elapsed time from when a job arrives for scheduling to when it completes execution, averaged over the entire workload. Response time includes both time spent in waiting queues and time spent in execution.

2. *average service time*: the elapsed time from when a job is allocated to compute nodes to when it is done with all its computation, communication and I/O, averaged over the entire workload. Execution time does not include time spent in waiting queues.

3. *average message blocking time*: the blocking time for one message (from one job) is the total time the message was blocked in router buffers along the route from the sending node to the destination node. The per job average message blocking time is the average blocking time for all of one job's messages.

4. *average interarrival time*: the elapsed time between two consecutive submissions of jobs to the system. This metric measures the offered load, and appears as the independent variable in the graphs of experimental results.

It is important to realize that *sustainable load*, the system load value below which average response time remains within *reasonable* bounds, is an important focal point in performance evaluation. This critical point is visible as the "knee" in the graphs of response time and system utilization (see Figure 19). Below the critical point, the system load is at manageable levels so that the increase in job response time is gradual and utilization continues to improve. At the knee, the job response time suddenly begins to grow rapidly toward infinity. By the same token, the system utilization levels off since the system is saturated with work. Thus, in evaluating the relative performance of resource management strategies, we focus our analysis on the phenomena observed near this saturation point.

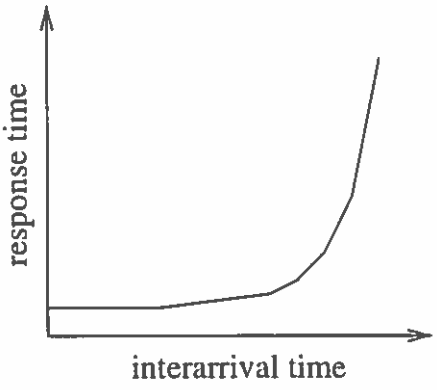


FIGURE 19. Sample Performance Graph

CHAPTER IV

ANALYZING COMMUNICATION-BASED NETWORK CONTENTION

Motivation

This chapter concentrates on communication-intensive jobs and investigates the relationship between *allocation compactness* and inter-job link contention.

As seen in Chapter II, non-contiguous processor allocation strategies assign nodes that are possibly dispersed, thus achieving vastly improved job throughput for computation-intensive workloads by eliminating fragmentation. However, earlier simulations [33] of the performance of several non-contiguous processor allocation strategies showed that although all non-contiguous strategies behave equally well in fragmentation experiments, their performance suffers in message-passing experiments where jobs execute communication patterns.

Figure 20 shows snapshots of several non-contiguous allocation strategies [33] servicing a given jobstream. All snapshots show the same jobs 11, 12 and 15. For the MBS strategy, the nodes allocated to jobs 11 and 12 are contiguous and the nodes allocated to job 15 form two convex rectangles. In contrast, the job allocations for the Paging strategy are "more dispersed", job 11 and job 12 consist of two clusters each and the nodes of job 15 are somewhat adjacent but do not form a convex shape at all. The job allocations for the Random strategy are even "more dispersed".

Differences in spatial layout obviously affect inter-job link contention. If in

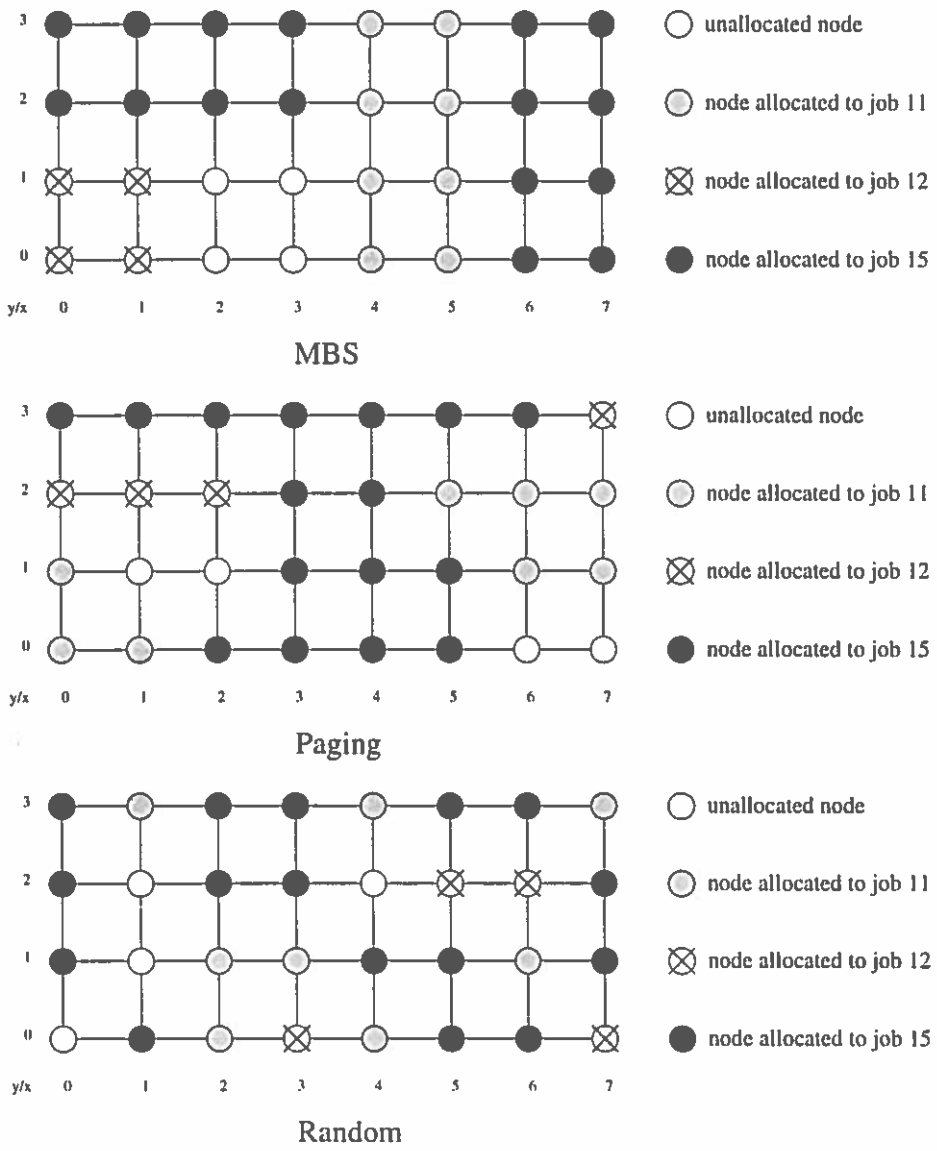


FIGURE 20. Snapshots of Different Non-Contiguous Strategies

Figure 21 the second node of job A had been (3,0) instead of (7,3), then there would have been no inter-job link contention.

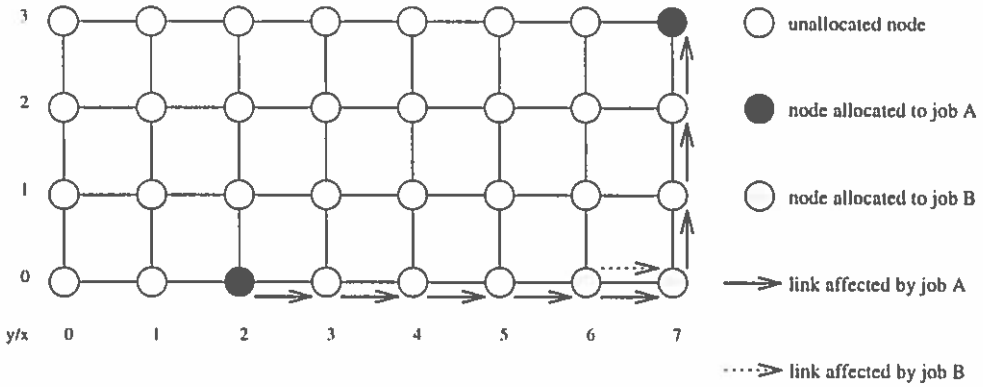


FIGURE 21. Inter-Job Link Contention for the Link Between (6,0) and (7,0)

Clearly, contiguous allocation strategies restrict the location of nodes allocated to a given job to form a convex shape and do not experience inter-job link contention, while non-contiguous allocation strategies do not restrict the location of the nodes allocated to a given job and do experience inter-job link contention. The problem we address is the degree to which non-contiguous strategies with "less dispersed" allocations are likely to experience less contention and how to define "less dispersed".

In this chapter, we define several *dispersal metrics* that measure the spatial layout of a given job's allocation in order to capture the degree of compactness of that allocation. We then study the degree of correlation between these metrics and inter-job link contention. Metrics that have high correlation with inter-job link contention and that are efficient to compute are useful for the evaluation and design of processor allocation algorithms. We run simulations using ProcSimity and compare dispersal metrics values for several known allocation strategies with

the corresponding contention measurements from the message-passing simulator. Our analysis and experiments considered different topologies of machines, a wide range of communication patterns and different workloads. Our results show that there is a very high correlation between degree of compactness as measured by our dispersal metrics and inter-job link contention. Thus, maximizing allocation compactness seems to be a very promising approach in order to minimize inter-job link contention.

Dispersal Metrics

Our first step is to develop a metric to quantify the degree of compactness of a given allocation. We start with "per job" dispersal metrics that are calculated on a per job basis. Per job dispersal metrics are based only on the addresses of the nodes allocated to the job in question. The addresses of those nodes are available at allocation time and are static over the lifetime of the job. Figure 22 shows a snapshot of an 8 x 4 mesh topology with three allocated jobs. This example is used throughout this section. We discuss three different categories of per job dispersal metrics as well as algorithms to compute them: *nodes_affected*, *links_affected* as well as distances and diameter. We consider two-dimensional mesh topologies first. Afterwards, we extend the algorithms so that they work for k-ary n-cube topologies as well.

Nodes Affected

Nodes_affected represents the number of nodes in the whole system that potentially suffer from inter-job link contention if the allocated nodes (i.e. the nodes

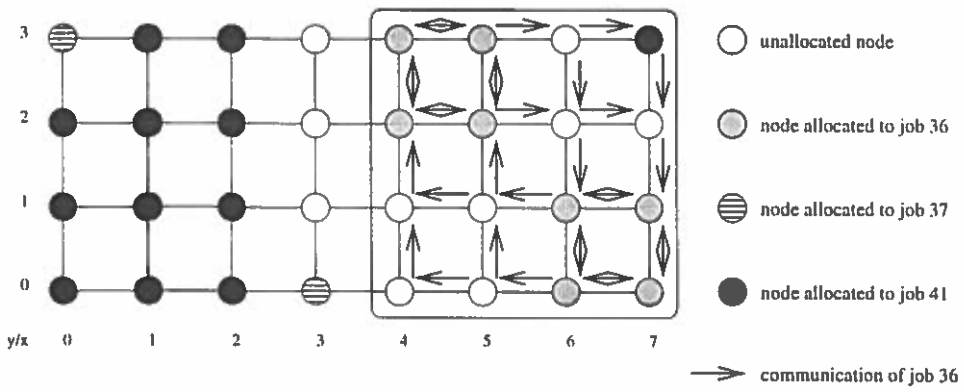


FIGURE 22. Enclosing Rectangle and Communication for Job 36

allocated to the job we look at) do all-to-all communication. This is indicative of contention because the more nodes are affected, the more likely is inter-job link contention. If a *foreign* node (i.e. a node not allocated to this job) is affected and it belongs to another job that does communication as well, messages from both jobs are likely to contend, resulting in inter-job link contention. Figure 22 shows the all-to-all communication of job 36. If foreign node (7,3) of job 41 wants to communicate with node (2,3) for example, inter-job link contention is likely to occur.

A metric that approximates the number of affected nodes (*nodes_affected*) finds the minimal enclosing rectangle that includes all the job's nodes and counts the number of nodes inside this rectangle. Let min_x and max_x be the minimum and maximum X coordinates among all nodes allocated to the job, and let min_y and max_y be the minimum and maximum Y coordinates among all nodes allocated to the job.

Definition: $nodes\_affected \equiv (max_x - min_x + 1) * (max_y - min_y + 1)$.

Figures 22 and 23 show the enclosing rectangle and the communication for

job 36 and 37, respectively. *Nodes_affected* for jobs 36, 37 and 41 is 16, 16 and 32, respectively. The enclosing rectangle for job 41 is the entire 8 x 4 mesh.

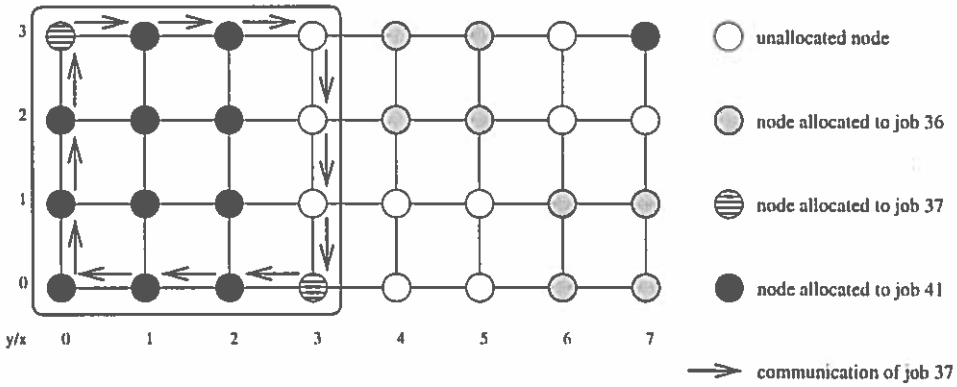


FIGURE 23. Enclosing Rectangle and Communication for Job 37

An efficient algorithm to calculate the enclosing rectangle examines each allocated node once to determine the minimal and maximal coordinates in each dimension. Thus, the algorithmic complexity for an n-dimensional mesh and job of size j nodes is $O(n * j)$.

Note that an exact count of the nodes affected only includes those nodes visited by messages sent according to XY-routing in the mesh. Thus, while the enclosing rectangle includes all nodes that are affected, it may also count nodes that are not affected. Figure 23 shows the minimum enclosing rectangle for job 37. Nodes (1,1), (1,2), (2,1) and (2,2) are inside the minimal enclosing rectangle, but they are not affected by communication of job 37. However, the complexity of an exact count is much higher: $O(n * j^2 * k)$ where k is the maximum number of nodes in one dimension. The worst case overestimate of *nodes_affected* occurs in a k x k mesh with one node allocated in the upper left corner and one node allocated in the lower right corner. In this case, *nodes_affected* is k^2 while the number of actual

nodes_affected is approximately $4k$.

Links Affected

The second metric we consider, *links_affected*, counts the number of links in the system that potentially suffer from inter-job link contention if the allocated nodes do an all-to-all communication. As defined above, let min_x and max_x be the minimum and maximum X coordinates among all nodes allocated to the job, and let min_y and max_y be the minimum and maximum Y coordinates among all nodes allocated to the job. Let $count_x$ be the number of distinct positions in the X dimension occupied by nodes assigned to the job; let $count_y$ be the number of distinct positions in the Y dimension occupied by nodes assigned to the job,

Definition: $links\_affected \equiv (max_x - min_x) * count_y + (max_y - min_y) * count_x$.

Figure 24 illustrates the computation of *links_affected* for job 41. $count_x$ and $count_y$ are computed by taking the projections of job 41's nodes onto the X and Y axes, respectively, yielding a value of $7*4+3*4 = 40$ *links_affected* for job 41. Note that while *nodes_affected* for jobs 36 and 37 are both equal to 16, the *links_affected* metrics differ for these two jobs with *links_affected* for job 36 equal to 24 and *links_affected* for job 37 equal to 12. Thus, *links_affected* is more discriminating and more accurately reflects the actual potential for inter-job link contention.

The complexity of an algorithm to compute *links_affected* for an n-dimensional mesh and job of size j nodes is $O(n * j)$ as well. There is no overestimate by *links_affected*: it computes exactly the number of links that are potentially visited by messages sent among the job's nodes according to XY-routing.

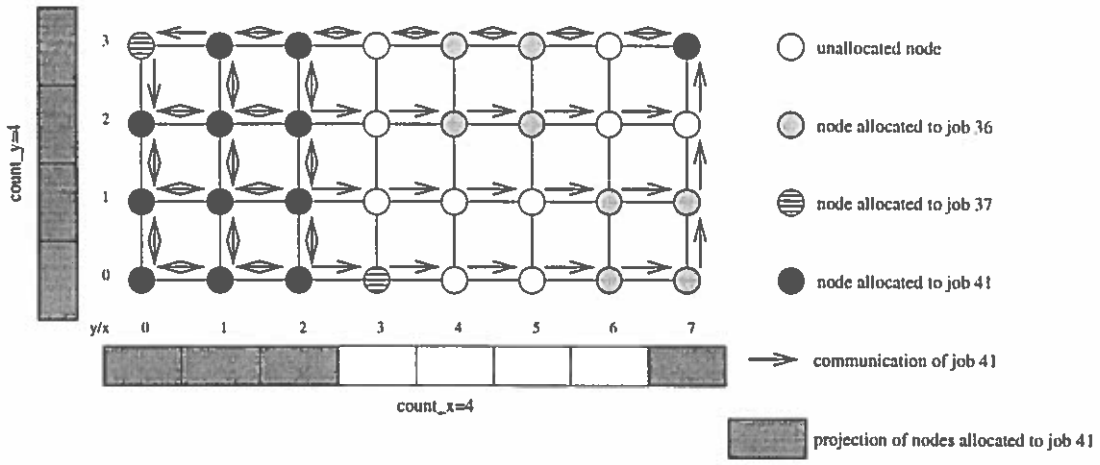


FIGURE 24. Bitvectors and Communication for Job 41

Distances and Diameter

We consider several possible dispersal metrics motivated by the field of cluster analysis [17].

1. *Average_distance* represents the average distance over all pairs of nodes allocated to the job.

2. *Summed_distance* represents the sum of the distances between all pairs of allocated nodes.

3. *Distance_from_center* represents the sum of the distances of each allocated node to the allocated node that is most central, i.e. the allocated node for which this sum is minimal.

4. *Diameter* represents the maximal distance between any two allocated nodes.

These metrics should correlate well with contention because the distance between two nodes equals the path length if the pair communicates using minimal dimension-ordered routing. The longer the path, the more likely is contention.

Let the nodes allocated to a given job be numbered arbitrarily from 1 to j, and let l and m be any two nodes with coordinates (l_x, l_y) and (m_x, m_y), respectively. Let $distance(l,m) = | (l_x - m_x) | + | (l_y - m_y) |$ be the Manhattan distance between nodes l and m.

Definition:

$$average\_distance \equiv \frac{\sum_{l=1}^{j} \sum_{m=1}^{j} distance(l,m)}{j * (j-1)}$$

$$summed\_distance \equiv \sum_{l=1}^{j} \sum_{m=1}^{j} distance(l,m)$$

$$distance\_from\_center \equiv \min_{l}(\sum_{m=1}^{j} distance(l,m))$$

$$diameter \equiv \max_{l,m}(distance(l,m))$$

The complexity of an algorithm to compute all of the above metrics is $O(n * j^2)$.

Extension to K-Ary N-Cube Topologies

While extensions to higher dimensional meshes are straightforward, some modifications are needed for k-ary n-cube topologies. K-ary n-cubes differ from n-dimensional meshes in that the k nodes in each dimension build a ring (the first and last node are connected through a "wrap-around" link). Dimension-ordered routing traverses one dimension at a time. To travel through one dimension, the message can travel in either direction (e.g. left or right in X-dimension) because

of the wrap-around. The direction taken depends on which one takes fewer hops.

The algorithm to compute *nodes_affected* and *links_affected* for meshes can be extended to k-ary n-cubes with minor changes. The complexity of the algorithm is $O(n*(j+k))$ where j is the number of nodes allocated to the job, n is the number of dimensions and k the arity of the k-ary n-cube. Whereas *links_affected* has no overestimate for meshes, marginal overestimates for k-ary n-cubes can occur, see Figure 25.

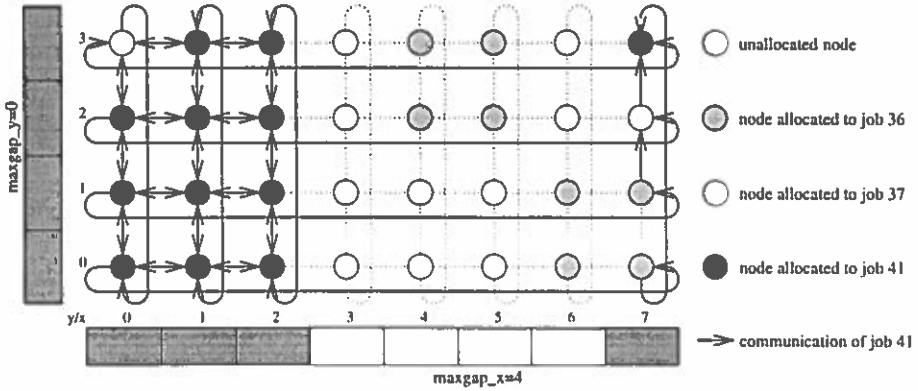


FIGURE 25. K-Ary N-Cube: Communication for Job 41

For our distance and diameter metrics, we use the notion of Lee distance [7] which extends the concept of Manhattan distance to n dimensions.

Definition:

$$lee\_distance(l,m) = \sum_{dim=1}^{n} min(\mid l_{dim} - m_{dim} \mid, k - \mid l_{dim} - m_{dim} \mid)$$

where $l = (l_1, l_2, ...l_n)$ and $m = (m_1, m_2, ...m_n)$.

The definitions of *average_distance*, *summed_distance*, *distance_from_center* and *diameter* for k-ary n-cubes are the same as those in the definition above, with $distance(l,m)$ being Lee distance rather than Manhattan distance. The algorith-

mic complexity remains at $O(n * j^2)$.

Average Dispersal

All the above dispersal metrics are computed for a single job ("per job dispersal metric"). *Average dispersal* is defined as the average value of a per job dispersal metric over a jobstream (workload) for a given allocation strategy ("dispersal metric per jobstream").

The following two scenarios show the problems exhibited by per job dispersal metrics and how they are resolved by average dispersal. Per job dispersal metrics are static, whereas the processor allocation problem in general and contention in particular are highly dynamic. Per job dispersal metrics do not consider other jobs, they are blind-folded so to speak:

1. To experience inter-job link contention, at least two different jobs have to interfere. A highly dispersed job A has a high value of per job dispersal metric. But if there is no other job to interfere with, inter-job link contention is zero.

2. If a dispersed job B interferes with a contiguous job C, both experience inter-job link contention but only B is likely to have a high value of per job dispersal metric.

One approach to resolve these problems is to take a snapshot and average the per job dispersal metrics over all jobs that are running at this point in time. This helps in the second scenario, because although the contiguous job C doesn't expect contention, the job B does.

In our experiments, we go a step further and average over all jobs of the jobstream, instead of taking many snapshots. This approach is more efficient and it also helps in the first scenario: it is very likely that there was another job in addition to job A a short time earlier or that there will be another job a short time later (in both simulation studies and the real world, machines normally have a high load) and contention was or will be occurring.

Impact of Allocation Compactness

The purpose of our experiments is to examine the degree to which our dispersal metrics correlate with inter-job link contention, and thus the ability of these metrics to be used to guide and evaluate processor allocation algorithms. We conduct two sets of experiments: the first set of experiments uses per job dispersal metrics. The second set of experiments uses average dispersal which considers the whole jobstream. Contention measurements are produced by a message-passing simulator. Our experiments consider both mesh and k-ary n-cube topologies of interconnection networks, a wide range of communication patterns and synthetic workloads.

Simulation Environment

To obtain contention measurements, the ProcSimity simulator described in Chapter III was used to simulate message-passing behavior down to the level of individual flits and message-passing buffers. Contention is measured by recording the amount of time the header flit of each message is blocked in the network waiting for a channel to become free.

Our simulation model uses wormhole switching and minimal dimension-ordered routing (XY routing for mesh and Lee routing for k-ary n-cube topologies). We model two uni-directional links between adjacent nodes and either a 16 x 32 mesh or a 8-ary 3-cube topology. To achieve a variety of spatial layouts, we employ the following allocation strategies. For mesh topologies we use the non-contiguous strategies Random [33], MBS [33] and Paging [33] (with different page sizes and different indexing schemes) as well as the contiguous strategies First Fit [58], Best Fit [58] and Frame Sliding [11]. For k-ary n-cube topologies we use the non-contiguous strategies Random [55], MBS [55], Paging [55] and Multipartner [55] as well as the contiguous strategies Buddy [32], Gray Code [8] and Partner [1]. The job scheduling policy is set to FCFS. We saw no significant differences when other job scheduling algorithms were used.

Workload

In our experiments, we use synthetic workload traces whose parameters for jobsize, service time, and interarrival time are based on realistic workload parameters from the Intel Paragon at the San Diego Supercomputing Center [56]. Our jobstreams consist of 1000 jobs. Jobsizes have an exponential distribution with mean of 16. For the purpose of non-empty waiting queues, we choose short interarrival times (Poisson distribution).

Five communication patterns are modeled: all-to-all broadcast, one-to-all broadcast, fast Fourier transform (FFT) as well as multigrid benchmark and kernel CG benchmark from NAS parallel benchmarks [4]. These cover many typical communication patterns and the complexity ranges from $O(j)$ to $O(j^2)$, j being

the jobsize. Figure 26 shows all-to-all and one-to-all communication for a job of size 4 as well as the different communication phases of FFT for a job of size 16. If the communication pattern has different phases, barrier synchronization is used between the phases.

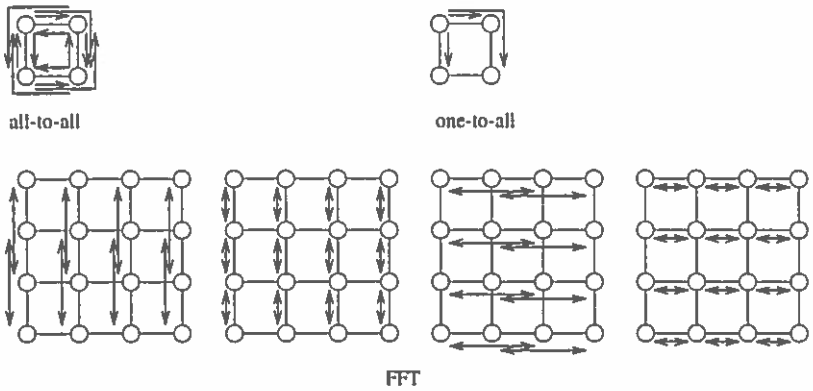


FIGURE 26. Communication Patterns

Since our focus is on inter-job link contention, we consider communication only (no computation). To study contention, we use heavy communication loads which are designed to produce heavy contention.

Results

Per Job Dispersal

The first set of experiments uses *per job dispersal* metrics. Given an allocation strategy and a communication pattern, we compute for each of the 1000 jobs of the jobstream the correlation between dispersal metric and contention. Contention is the average blocking time of messages of that job as measured by the simulator.

For correlation computations, we use the Pearson correlation coefficient. Results reported represent the statistical mean after 20 simulation runs with identical

parameters, and given 95% confidence level, mean results have less than 3% error.

Overall, correlations of per job dispersal metrics with inter-job link contention were not strong. Table 1 gives results representative of the full set of experiments which ranged over six allocation strategies and five communication patterns. Table 1 shows the correlations for a 16 x 32 mesh topology, MBS allocation strategy and two different communication patterns. Figure 27 shows the scatter plot for all-to-all communication and the dispersal metric *nodes_affected*.

TABLE 1. Correlation for Per Job Dispersal Metrics (MBS Strategy, 16x32 Mesh)

	all-to-all	one-to-all
nodes_affected	.391583	.256183
links_affected	.444805	.225672
average_distance	.407117	.499982
distance_from_center	.499775	.279724
summed_distance	.501765	.507082
diameter	.361437	.222952

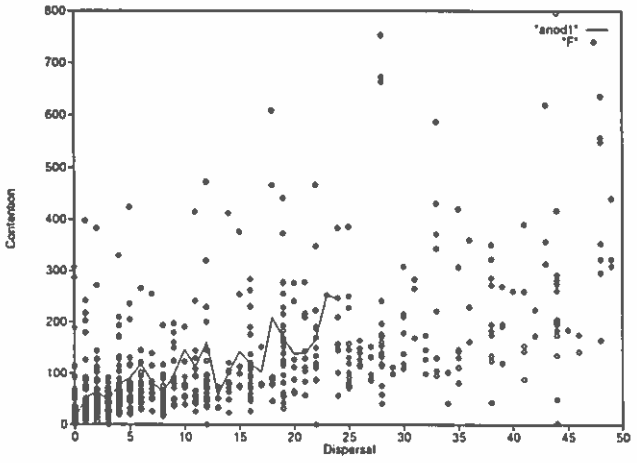


FIGURE 27. Scatter Plot of Per Job Dispersal Metric (*Nodes_Affected*) vs. Contention (All-to-All Communication, MBS Allocation Strategy, 16x32 Mesh)

Correlation ranges from 0.22 to 0.50 and the points of the scatter graph do

not fall in a straight line. The lack of strong correlations are due to the fact that per job dispersal metrics ignore the presence of other jobs in the system while inter-job link contention depends precisely on interference from other jobs. Therefore, *per job dispersal* of two jobs can be very different, while they experience the same level of inter-job link contention.

To detect any possible trends, we group the datapoints with similar dispersal values into intervals, compute the average contention for each interval and connect these points. The resulting curve is shown in Figure 27, which seems to indicate that in general as dispersal increases, contention also increases.

Different communication patterns and different allocation strategies result in graphs very similar to Fig 27. The same is true for k-ary n-cube topologies.

Average Dispersal

The second set of experiments uses *average dispersal* (per jobstream). As discussed earlier, they resolve the problem of per job dispersal metrics being blind to the presence of other jobs in the system.

Table 2 gives results representative of the full set of experiments. It shows the correlation for an 16 x 32 mesh topology and five different communication patterns. Table 3 shows the correlations for a 8-ary 3-cube and two different communication patterns. The correlations are very high, varying between 0.890 and 0.998 overall. For both topologies the dispersal metric *summed_distance* has the highest correlation for all-to-all communication. For other communication patterns in mesh topologies, the dispersal metric *diameter* achieves the highest correlation. For one-to-all communication in the k-ary n-cube topology, the dispersal metric *links_affected* has the highest value.

TABLE 2. Correlation for Average Dispersal (16x32 Mesh)

	all-to-all	one-to-all	FFT	MultiGrid	NASCG
nodes_affected	.910878	.996691	.985869	.986541	.992155
links_affected	.927287	.997315	.993006	.989280	.992311
average_distance	.913628	.998490	.990213	.991278	.995726
distance_from_center	.979404	.973173	.980911	.960587	.959212
summed_distance	.996203	.925024	.934681	.895492	.890912
diameter	.939838	.996631	.995940	.994470	.996535

TABLE 3. Correlation for Average Dispersal (8-Ary 3-Cube)

	all-to-all	one-to-all
nodes_affected	.959740	.990280
links_affected	.942668	.993962
average_distance	.953251	.991443
distance_from_center	.973884	.964315
summed_distance	.981271	.936519
diameter	.971608	.964080

Figure 28 shows the scatter plot for *average diameter* and for all five communication patterns in one graph. For each communication pattern, the datapoints are close to a straight line. This visually shows the high correlations. The slopes of the regression lines vary for different communication patterns. All-to-all communication has the biggest slope and one-to-all shows the smallest slope. Other average dispersal metrics have similar graphs.

Figure 29 shows the scatter plot for FFT communication, for *average diameter* and for 20 jobstreams that differ in the seeds for the probabilistic parameters. It shows four clusters, the lowest for the contiguous strategies First Fit, Best Fit and Frame Sliding, the second for MBS (non-contiguous), the third for Paging (non-contiguous) and the fourth for Random (non-contiguous) allocation strategy.

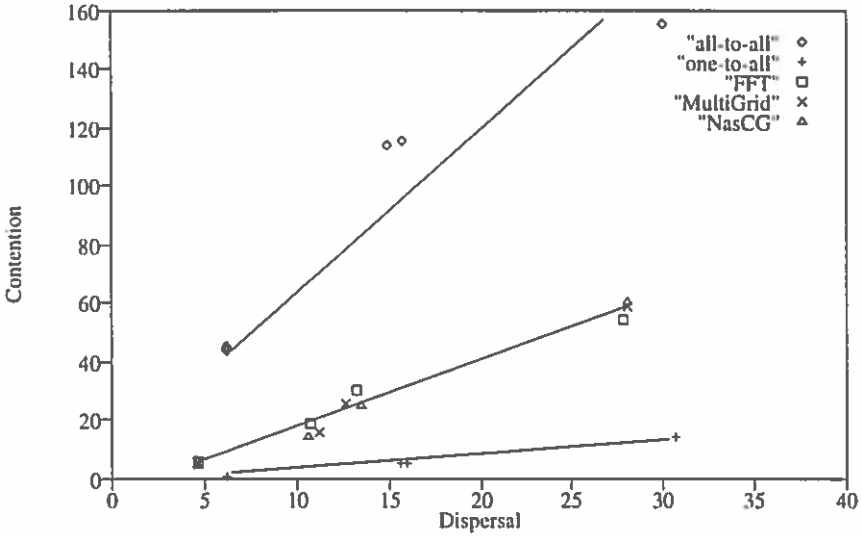


FIGURE 28. Scatter Plot for Five Communication Patterns (Average Dispersal Metric *Diameter*, 16x32 mesh)

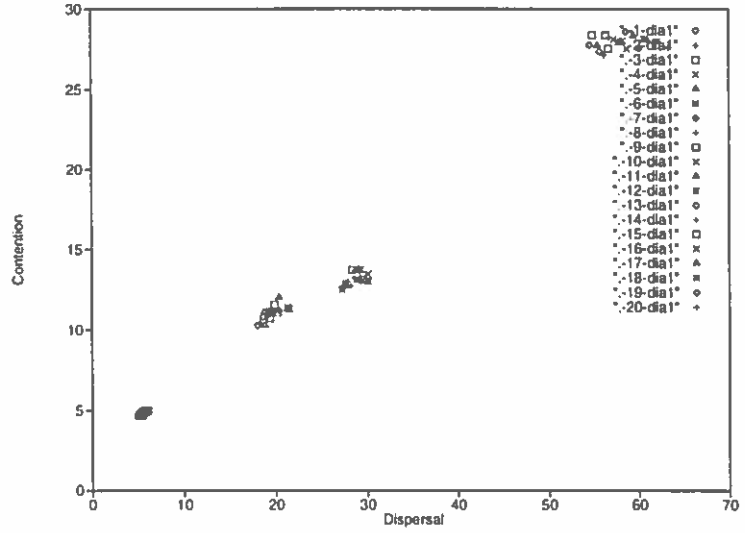


FIGURE 29. Scatter Plot for 20 Jobstreams (Average Dispersal Metric *Diameter*, 16x32 Mesh, FFT Communication)

In this graph, the slope of the regression line for different workloads is almost identical.

Figure 30 shows rank correlation: the example chosen is all-to-all communication pattern and the average dispersal metric *links_affected*. Dispersal metric, measured contention and measured average service time rank the different allocation strategies in the same order. (First Fit, Best Fit and Frame Sliding have the lowest values, followed by Paging and MBS, Random is last.) Since dispersal metrics are much easier to obtain, they have the potential to replace costly low-level simulations as a means for evaluating allocation algorithm performance.

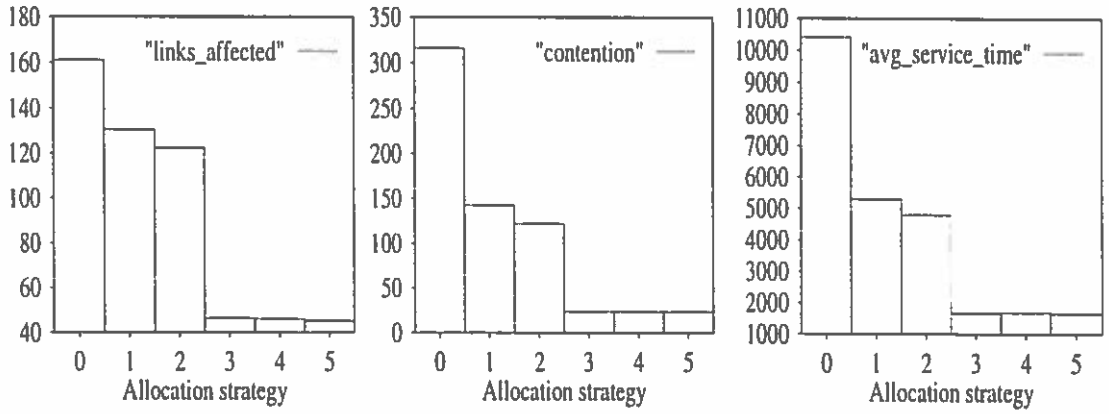


FIGURE 30. Rank Correlation for Six Allocation Strategies (Average Dispersal Metric *Links_Affected*, 16x32 Mesh, All-to-All Communication)

In additional experiments, we reduce the average number of messages sent per job. Much less contention is measured. This results in lower correlations (0.08 to 0.61) and indicates there is no strong relation between dispersal and contention if there is only little contention. It is easy to see that dispersal metrics are still discriminating among allocation strategies, even if the latter experience almost the same low contention.

Discussion

Non-contiguous processor allocation strategies assign nodes that are possibly dispersed, thus achieving vastly improved job throughput for computation intensive workloads. For further improvement, inter-job link contention due to communication must be minimized as well. Our contribution towards this goal is (1) an analysis of the relationship between inter-job link contention and allocation compactness and (2) a set of dispersal metrics that measure allocation compactness. Our six dispersal metrics are *nodes_affected*, *links_affected*, *average_distance*, *summed_distance*, *distance_from_center* and *diameter*.

To summarize our results:

1. Maximizing allocation compactness seems to be a very promising approach in order to minimize inter-job link contention for communication-intensive jobs. Our simulation experiments show that strategies that allocate more compactly (as measured by our dispersal metrics) experience less inter-job link contention. Correlations of average dispersal metrics (over a workload) with contention measured by the simulator range from 0.890 to 0.998 for a wide range of communication patterns and two network topologies (mesh and k-ary n-cubes).

2. Average dispersal metrics have the potential to help evaluate non-contiguous processor allocation strategies. Our dispersal metrics are efficient to implement for mesh and k-ary n-cube interconnection topologies. Because they are efficient to compute and have high correlation with contention, average dispersal metrics can be used to evaluate allocation strategies and thus in many cases replace costly low-level simulations.

3. Per job dispersal metrics have the potential to help improve non-contiguous processor allocation strategies. Selecting the job allocation with minimal contention is necessary to improve non-contiguous allocation strategies. Because per job dispersal metrics can be efficiently computed at allocation time and because they contribute to average dispersal, per job dispersal metrics can be used within non-contiguous allocation strategies to help minimize communication-based contention and thus optimize overall performance.

CHAPTER V

MINIMIZING COMMUNICATION-BASED NETWORK CONTENTION

Motivation

To further improve the performance of non-contiguous processor allocation (that overcome the severe fragmentation problem), we develop a new allocation strategy that aims at minimizing communication-based network contention. Motivated by our observations about allocation compactness (see Chapter IV), our new algorithm tries to assign to each new job a cluster of nodes that is as compact as possible.

The MC Allocation Strategy

Our new allocation algorithm is called MC due to its mission of Minimizing inter-job link Contention.

We first discuss MC as applied to two-dimensional meshes. On these machines, job requests are typically specified by two numbers, width w and height h. The MC algorithm begins by constructing several candidate clusters, one per idle node (i, j). A candidate cluster is centered around (i, j) and contains idle nodes arranged within *shells* radiating from the center node. For the purpose of defining shells, we disregard whether nodes are idle (unallocated) or busy (allocated to other jobs).

Definition: For a given center node (i, j) and a request size $w \times h$, $shell_0$ is

the contiguous $w \times h$ rectangle centered around node (i, j), i.e. the $w \times h$ rectangle whose lower-left corner is the node $(i - \lceil \frac{w-1}{2} \rceil, j - \lceil \frac{h-1}{2} \rceil)$. $Shell_{s+1}$, $s \geq 0$ are successive rectangular rings of nodes. More precisely, $shell_{s+1}$ contains the nodes that are at distance 1 from at least one node in $shell_s$, but not contained in any $shell_k$, $0 \leq k \leq s$, where distance is defined as $max(\Delta x, \Delta y)$.

To construct a candidate cluster, idle nodes are selected from successive shells, beginning with $shell_0$, until enough (i.e. $w*h$) idle nodes have been selected to satisfy the request. The order in which idle nodes within each shell are selected is designed to find a rectangle of dimension $w \times h$ if it exists or to find a compact and as square as possible cluster otherwise. Thus, idle nodes on the shorter sides of the ring are selected first, idle nodes on the longer side second, and corner nodes last.

To evaluate a candidate cluster, we define cost in the following way.

Definition: The cost of a node is equal to the value of that node's shell number. The cost of a candidate cluster is the sum of the costs of the nodes in that cluster.

Since a $w \times h$ request can be equivalently satisfied by a $w \times h$ or a $h \times w$ allocation, MC considers both orientations and evaluates two candidate clusters for each idle node. In the end, the candidate clusters with minimal total cost from both orientations are compared. Pseudocode for the algorithm is given in Figure 31.

We illustrate the algorithm given the situation in Figure 32 and a job request of 1×3. Each of the idle nodes (4 in our case) does a shell-like scan to find a compact cluster of 3 nodes with itself as the center. Figure 32 shows the shells for

```
mc_allocate(w, h){
  if number_idle < w * h then return fail
  for both orientations w x h and h x w
    for each idle node (i, j)
      cluster = empty list; tcost = 0
      add idle nodes to cluster that are in
        w x h rectangle around (i, j) /* shell_0 */
      for each shell s >= 1
        stop if |cluster| = w * h
        if node idle then add to cluster; tcost += s
    select cluster with minimal tcost
  compare best cluster from both orientations
  allocate those nodes, update number_idle and busy array
}
```

FIGURE 31. MC Pseudocode

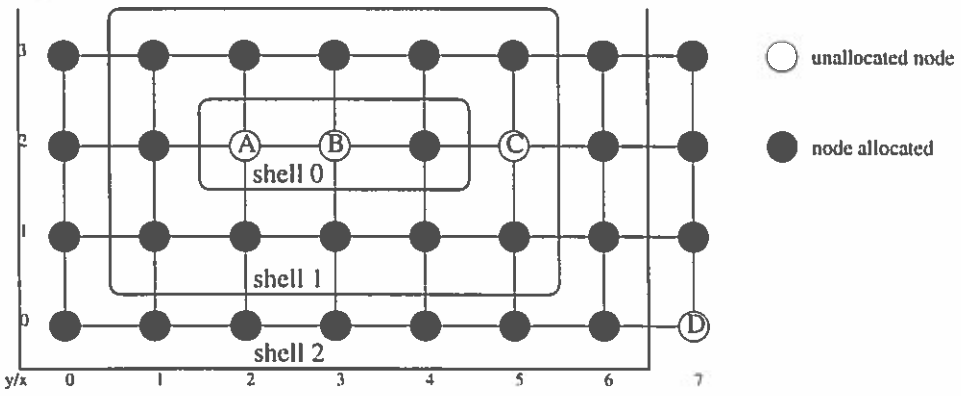


FIGURE 32. Shells Around B, Request of 1×3

idle node B. $Shell_0$ is the 1×3 block (the exact job request size and shape) around node B. However, only two nodes are idle in $shell_0$, thus MC keeps searching through bigger shells until we have found enough idle nodes. $Shell_1$ extends $shell_0$ by one ring of nodes, and so on. In the end, the candidate cluster centered around node B includes the idle nodes A, B and C for a total cost of 1. Similarly, the candidate clusters centered around nodes A, C and D have total cost of 2, 3 and 5, respectively. The cluster centered around node B has minimal cost and will be allocated.

MC is distinguished from previous fragmentation-free strategies by several characteristics: it finds compact clusters, it considers single nodes instead of blocks, and it is inherently parallelizable. We elaborate on each of these features below.

Property: MC finds compact clusters thereby minimizing inter-job link contention. If a contiguous allocation of requested size exists, MC finds it. Otherwise, MC yields a compact allocation, never leaving idle nodes unselected on communication paths between selected nodes. This is a result of the shell-like scanning scheme centered around each idle node and of the definition of cost. Figure 33 shows how previous strategies (MBS, M2DB and AS&MB) assign a 2×4 request. The allocation by MC (indicated by shells in Figure 33) causes significantly less inter-job link contention and the new job does not interfere with jobs B or C.

Property: MC builds clusters out of individual nodes and thus circumvents restrictions imposed by previous block-based approaches. These block-based algorithms allocate in a "conservative" manner: larger blocks or subrequests are not broken down further if the job can be allocated without doing so. There are two problems with such schemes: First, acting conservatively does not help the

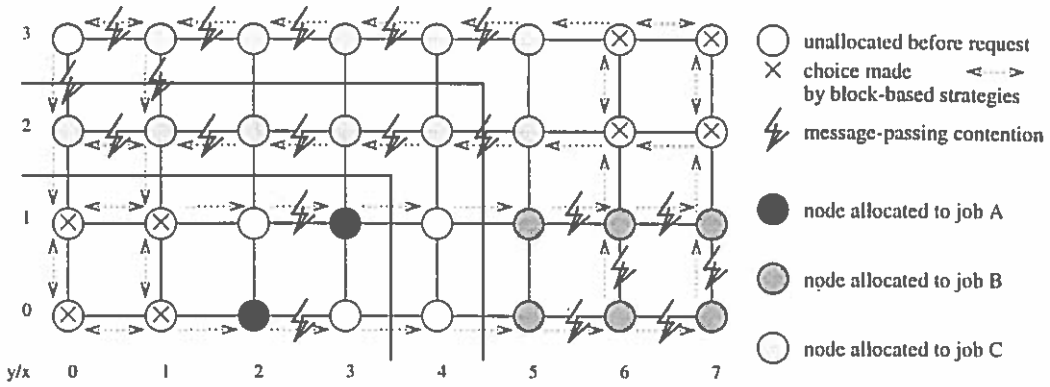


FIGURE 33. Block-Based Strategies, Request of 2×4

allocation of future jobs if either the preserved block will be split up anyway or if the preserved block will not be used anyway. The first situation arises if the next job needs smaller blocks. The second situation arises if the next job has to wait until enough idle nodes are available (yielding a new situation) or if no next job arrived yet. More important, acting conservatively often results in less compact allocations. Given the situation in Figure 34, a request of 3×2, MC yields a better solution than MBS and M2DB: it considers single nodes and constructs a compact cluster from scratch.

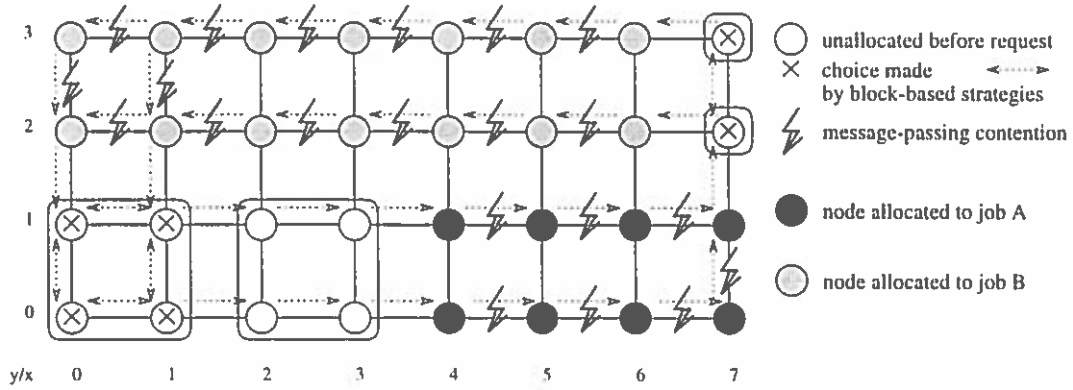


FIGURE 34. Allocation of a 3×2 Request by Block-Based Strategies and Resulting Inter-Job Link Contention

Property: MC is inherently parallelizable. Each idle node can independently find and evaluate the candidate cluster of which it is the center. To allocate a new job, a distinguished system node sends the busy array, a bitvector representing the status of each node, to each idle node and—after collecting the total cost (and node selections) of the candidate clusters—allocates the job to the best candidate cluster. Thus, we employ just the idle nodes and need only limited communication in the form of a scatter-gather pattern. Because interarrival times of jobs are typically on the order of minutes and runtimes of jobs are on the order of hours, the additional resource usage of a parallel implementation of MC would be negligible.

Further characteristics of MC are: it takes the aspect ratio of requests into account because $shell_0$ has the requested shape. Moreover, MC easily extends to higher dimensional meshes and toroidal topologies; extension of the definition of shells and the shell-like scan are straightforward.

Performance Evaluation

We conducted experiments to compare MC to previous non-contiguous allocation strategies. We again use ProcSimity to model a mesh architecture with wormhole switching, minimal dimension-ordered routing, and two uni-directional links between adjacent nodes. The detailed message-passing behavior is simulated down to the level of individual flits and message-passing buffers. In the results reported, we model a 16×22 mesh and FCFS scheduling. We compare the performance of MC with three non-contiguous allocation algorithms: MBS [35], Paging [35], and AS&MB [52]. These algorithms have been shown to have the best performance among all non-contiguous strategies to date. We measure two high-level

performance metrics, response time and service time, and two contention-related metrics, blocking time and *nodes_affected*. Blocking time records the amount of time the header flit of each message is blocked in the network waiting for a channel to become free; *nodes_affected* as defined in Chapter IV was chosen to measure the degree of a job's dispersal since it was easy to compute and its correlation to link contention was very close to +1.

To allow for both easy comparison with previous experiments [33, 52] and for a realistic evaluation of MC, our performance evaluation includes the use of three different workloads: one synthetic workload and two trace-derived workloads. Workload I is a synthetic workload, a stream of 1000 jobs whose sizes are exponentially distributed. Workload II and III, described in detail below, are both trace-derived workloads, a stream of 6087 real production jobs recorded over a three month period from October 1 to December 31, 1996, from the Intel Paragon at the San Diego Supercomputer Center. The traced job stream is taken only from the 352 node NQS partition [56] of the machine, through which all batch jobs were scheduled. The trace had the following statistical characteristics: the mean interarrival times was 1301 seconds, with a coefficient of variance of 3.7; the average job size was 14.5 nodes, with a coefficient of variance of 1.5, and with the distribution heavily favoring sizes that are powers of two.

Our trace-derived workload uses the empirical arrival times and job sizes. To challenge allocation strategies, we multiply job arrival times by constant factors c to create system utilizations in the range of about 10% to about 90%. When $c < 1$, simulated interarrival times decrease, resulting in increased system load. Since our focus is on inter-job link contention, we model a heavy communication

load and no computation. Each job does all-to-all communication, i.e. each node sends a message to all other nodes of the same job. In workload II, number of messages sent and jobsize are correlated since each job does exactly one iteration of the all-to-all communication. In workload I and III the number of messages sent is uncorrelated to the jobsize. We use an exponential distribution and set the mean such that a job of average size completes one iteration of an all-to-all communication.

As shown in Figure 35a, average response time of jobs is much lower if MC is used instead of MBS, Paging or AS&MB. The maximum reduction in average response time is 86.25% for workload I, 81.35% for workload II (at interarrival time 900) and 97.06% for workload III.

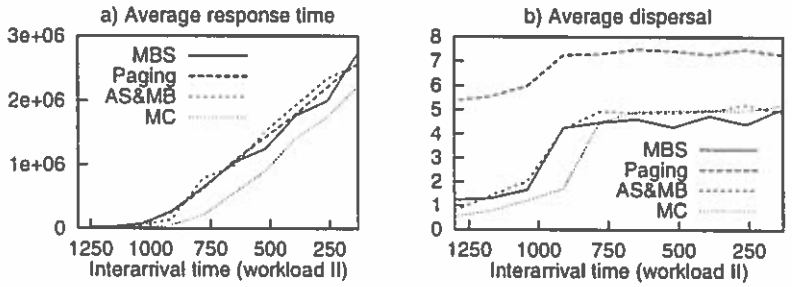


FIGURE 35. Average Response Time and Average Dispersal

Figures 35 and 36 help to explain the outstanding performance of MC: up to a certain interarrival time (800 in case of workload II), MC succeeds at allocating jobs very compactly (as measured by dispersal in Figure 35b) and thus at keeping contention low (Figure 36a). If messages contend less, jobs can leave the system earlier (Figure 36b). This in turn decreases the waiting time of future jobs or it makes it easier to allocate future jobs compactly.

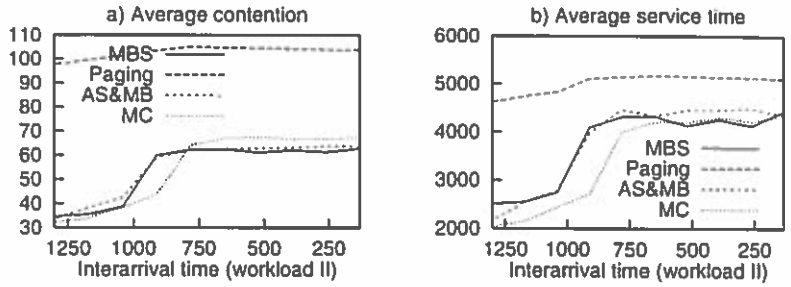


FIGURE 36. Average Contention and Average Service Time

The sharp increase in Figure 36b is critical: Although utilization of the system increases from about 85% to about 90%, average service time doubles thus hurting throughput severely. Clearly, the system should not be operated at such high loads, regardless of allocation strategy. However, using MC the system can sustain higher loads before this sharp increase occurs. Below this load (interarrival time of 800 in case of workload II) MC achieves better dispersal and contention values and thus outperforms MBS, Paging and AS&MB.

Conclusions

Taking on the challenge of reducing inter-job link contention in fragmentation-free processor allocation, we designed and tested a new allocation strategy called MC. To summarize our results:

1. MC is a simple, elegant and efficient strategy. Each idle node builds a cluster with itself as the center, using a shell-like scanning scheme and taking the aspect ratio of the request into account.

2. MC is inherently parallelizable, employing just the idle nodes and needing only limited communication in the form of a scatter-gather pattern.

3. MC finds a contiguous rectangle of requested size if one exists or finds a compact cluster otherwise. Compact allocations help minimizing inter-job link contention, in agreement with the results of our analysis (see Chapter IV).

4. MC outperforms previous allocation strategies, reducing average response time up to 97% (when communication costs dominate runtimes) and being capable of sustaining higher system loads. These performance results are based on message-passing simulations using workload traces from the San Diego Supercomputing Center.

CHAPTER VI

ANALYZING I/O-BASED NETWORK CONTENTION

Motivation

In this chapter, we analyze traffic hotspots due to data transfer between compute nodes and I/O nodes. Our goal is to study the effects of spatial layout of jobs on I/O-based network contention and parallel I/O performance. This study is motivated by earlier I/O measurements on a TFLOPS machine with up to nine I/O nodes and several hundred processor nodes at Intel [22, 24]. These experiments showed that certain placements of compute nodes and I/O nodes saturated the bandwidth of some network links, resulting in serious parallel I/O performance degradation [23].

In order to understand the phenomena observed, we conduct a careful analysis involving both dynamic simulation and analytic modeling. We evaluate the parallel I/O performance sensitivity to number of I/O nodes, number of compute nodes, throughput rate per network link, throughput rate per I/O node, spatial layout of allocated compute nodes, and read or write traffic. We assume a TFLOPS-like architecture with a mesh-based topology, XY-wormhole-routing, and the I/O nodes configured on one side of the mesh.

Our study yields surprisingly strong results regarding (1) the limitations on parallel I/O performance due to network contention, and (2) the possible gains in parallel I/O performance that can be achieved by tuning the spatial layout of jobs.

By controlling both the shape of the compute nodes allocated to jobs and their location relative to the I/O nodes, we can significantly reduce network contention and thus improve parallel I/O performance.

Figure 37 illustrates how the spatial layout of jobs affects parallel I/O performance. In this simple example, one job runs on two compute nodes and writes data to two I/O nodes as shown in Figure 37a. Assuming that throughput rate per I/O node equals throughput rate per network link, it turns out that overall parallel I/O performance for layout 1 (shown in Figure 37b) is only one half that of layout 2 (shown in Figure 37c). This difference in parallel I/O performance is due to network contention incurred by data in transit to the I/O nodes.

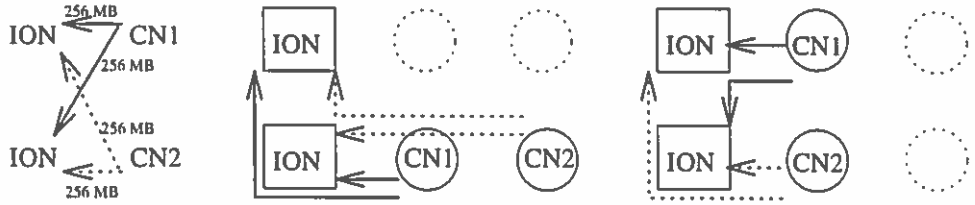


FIGURE 37. Simple Example Showing That Spatial Layout Affects I/O Throughput

Let us assume that each compute node (CN) sends 256 MB of data to each I/O node (ION), and that both throughput rate per I/O node and throughput rate per network link per direction is 64 MB/s. The approximate time it takes each I/O node to write its 512 MB of data to disk is $\frac{512MB}{64MB/s} = 8$ seconds. However, if the job is allocated as shown in Figure 37b, data from all four sender-receiver pairs (1 GB) has to traverse the link that is located between CN1 and the lower I/O node in west-bound direction. This takes $\frac{1GB}{64MB/s} = 16$ seconds. Thus, the network

limits the overall parallel I/O performance to $\frac{1GB}{16s} = 64$ MB/s. In contrast, if the job is allocated as shown in Figure 37c, every network link carries at most 512 MB of data per direction. Now, the network can accomplish the data transfer in $\frac{512MB}{64MB/s} = 8$ seconds. This matches the time it takes the I/O nodes to write the data to disk, yielding overall parallel I/O performance of $\frac{1GB}{8s} = 128$ MB/s, which is twice as big as the overall parallel I/O performance of the other layout.

We see the same effect on larger systems, even if f – the ratio of throughput rate per network link and throughput rate per I/O node – is greater than one. Figure 38 shows simulation results for $f = 6$ and one job that runs on 400 compute nodes and writes data to the I/O nodes. When varying the number of I/O nodes from 2 to 32, parallel I/O performance would ideally grow linearly with the number of I/O nodes. However, Figure 38 shows two things: parallel I/O performance levels off (due to network contention), and spatial layout of jobs makes a significant difference. 1200 MB/s of parallel I/O performance can be achieved with layout 2, compared to only about 600 MB/s with layout 1.

Analytic Modeling

Two characteristics define a job's spatial layout: the "shape" of the allocated compute nodes and their location within the mesh topology. The approaches used by current commercial allocation software fall into two categories: block-based allocation schemes [53] or linear allocation [41, 45]. The five spatial layouts that we consider in this study are generalizations of these known allocation schemes. They are block corner, block center, orthogonal, parallel, and diagonal (see Figure 39).

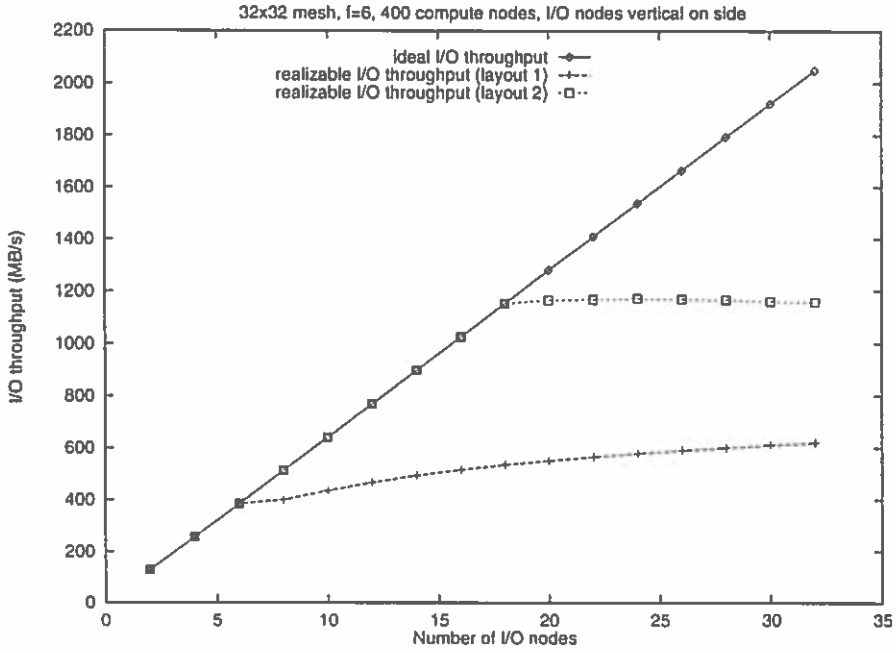


FIGURE 38. Parallel I/O Throughput Degradation

We look at several performance metrics. *Max_contention* measures the maximum number of I/O traffic sender-receiver pairs that contend for the same network link. We look at throughput for the I/O nodes alone, throughput for the network alone in the presence of contention, and the *realizable I/O throughput rate* which is the minimum of I/O node and network throughput. Our analytic model assumes no blocking overhead.

Graph Theoretic Model

To understand the nature of network contention arising from parallel I/O traffic and its effect on parallel I/O performance, we refine the graph theoretic model described in Chapter III to analyze the effects of spatial layout of jobs on network contention and parallel I/O performance.

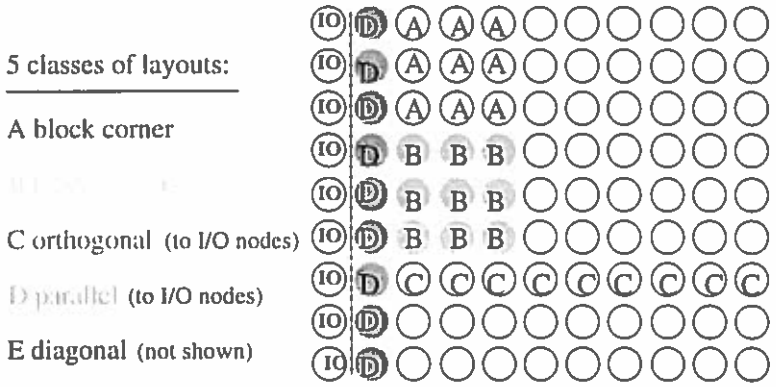


FIGURE 39. Five Classes of Spatial Layout of One Job of Size 9

All symbols and terms are summarized in Table 4.

TABLE 4. Notation and Definitions

m	number of I/O nodes
n	number of compute nodes exchanging data
k	length of one side of the quadratic mesh of compute nodes of the machine
B_0	effective throughput rate per I/O node
B_l	effective throughput rate per network link
f	quotient of B_l and B_0
link_contention	number of distinct sender-receiver pairs using that link
hotspot	link with maximal *link_contention*
max_contention	value of *link_contention* at hotspot
B_{ION}	I/O node throughput, I/O throughput of all I/O nodes
B_{NET}	network throughput, throughput of the network considering the limiting hotspot link
realizable throughput	min(B_{ION}, B_{NET})

Definition: The *system architecture graph* consists of m I/O processors located vertically on the west side of a $k \times k$ mesh of compute processors.

A given parallel application requires n compute nodes to be assigned to processors of the mesh. As described in Chapter III, we assume the I/O requirements

of a job to be that of personalized broadcast, from all m I/O nodes to all n compute nodes (allocated to the job) in the case of read, and from all n compute nodes to all m I/O nodes in the case of write.

Definition: The *I/O traffic graph* $K_{m,n}$ is a complete bipartite graph with one set of vertices representing I/O nodes and the other set representing compute nodes. Edges are directed from the set of I/O nodes to the set of compute nodes in the case of a *read* traffic graph.

We *embed* the graph $K_{m,n}$ into our architecture graph subject to the following constraints:

1. the m I/O nodes of the I/O traffic graph are mapped to the m I/O processors in the architecture graph;

2. the n compute nodes of the I/O traffic graph are mapped to some subset of the compute processors in the mesh; we will consider five possible mappings that correspond to the five spatial layout schemes described above (block corner, block center, parallel, diagonal, orthogonal).

3. each edge from I/O node i to compute node j in the I/O traffic graph is mapped to the path from I/O processor $a(i)$ to compute processor $a(j)$ according to the XY-routing scheme of the mesh.

Definition: The *link contention* of a link l is the number of edges of the I/O traffic graph whose corresponding paths include l when we embed the I/O traffic graph into the mesh network as described above. Conceptually, link contention is the number of distinct sender-receiver pairs sharing that network link.

Definition: A link with maximal link contention is a *hotspot* and the value of

this maximum contention is denoted by *max_contention*.

Referring back to the example of Figure 37, the four sender-receiver pairs are mapped to different, but overlapping, paths in the network. This results in *max_contention* of 4 for layout 1 and *max_contention* of 2 for layout 2.

Definition: The *I/O node throughput rate* is the rate at which all I/O nodes combined can read data from or write data to disks. The interconnection network between nodes is ignored. The I/O node throughput rate is the aggregate of all throughput rates per I/O node, B_0. Thus,

$$B_{ION} = m * B_0$$

Definition: The *network throughput rate* is defined as the aggregate data transfer rate through the interconnection network and is equal to the total amount of data to be transfered divided by the time incurred by the bottleneck link to transfer all the data of its *max_contention* sender-receiver pairs. Assuming that each message carries D bytes of data, the total amount of data to be transfered is $m * n * D$, and the time incurred by the bottleneck link is $\frac{max\_contention * D}{B_l}$. Thus, the network throughput rate is

$$B_{NET} = \frac{m * n * B_l}{max\_contention}$$

Definition: The *realizable I/O throughput rate* is the rate at which I/O data can be transfered between compute nodes and disks. It is equal to the lesser of the

I/O node throughput rate and the network throughput rate.

$$realizable\ I/O\ throughput = min(B_{ION}, B_{NET})$$

Best and Worst Case Network Contention and Network Throughput

In this section we develop formulas for the theoretical highest (worst) and lowest (best) levels of *max_contention* possible in our system. These values can be used to determine the theoretical worst and best network throughput rates. For the following theorems recall that we assume mesh architectures, XY-routing, complete bipartite I/O traffic graphs and I/O nodes configured on one side of the mesh.

Theorem 1: The worst and best case values for *max_contention* are $m * n$ and $\frac{m*n}{4}$, respectively.

Proof: Given XY-routing, the level of contention on any given link can be computed by straightforward formulas. For example, the link contention on any west-bound channel from $(x+1, y)$ to (x, y) is given by the number of senders (u, v) with $u > x$ and $v = y$ times the number of receivers (s, t) with $s \leq x$.

Recall the definitions for *max_contention* and *link_contention*. Given $m * n$ sender-receiver pairs, in the worst case all of them share one link. Thus, $max\_contention \leq m * n$. This worst case value of $m * n$ is achievable, for example when the compute nodes lie orthogonally in relation to the I/O nodes as shown in Figure 40a. In this spatial layout, all $m * n$ sender-receiver pairs contend for the horizontal hotspot link.

We prove that the best case value for *max_contention* is $\frac{m*n}{4}$ by (1) prov-

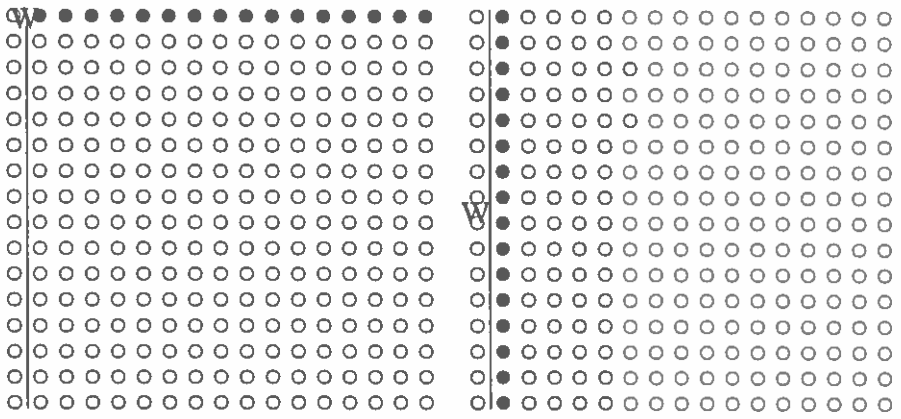


FIGURE 40. Two Spatial Layouts That Achieve the Worst Case and the Best Case Value for *Max_Contention*

ing that for all possible layouts, *link_contention* on a particular link, the middle I/O link, cannot be less than $\frac{m*n}{4}$, and (2) showing a layout that achieves $max\_contention = \frac{m*n}{4}$. Since *max_contention* is the maximal *link_contention* over all links, including the middle I/O link, $max\_contention \geq \frac{m*n}{4}$.

Link_contention in either the south-bound or north-bound direction of the link in the middle of the I/O nodes cannot be less than $\frac{m*n}{4}$. The link contention in south-bound direction from node $(x, y+1)$ to node (x, y) is given by the number of senders (u, v) with $v > y$ times the number of receivers (s, t) with $t \leq y$ and $s = x$. The south-bound direction of the link in the middle of the I/O nodes has $\frac{m}{2}$ receivers, independent of the spatial layout of the job, and i senders, $0 \leq i \leq n$, if i compute nodes are allocated in rows above the middle of the I/O nodes. Similarly, the north-bound direction of the link in the middle of the I/O nodes has $\frac{m}{2}$ receivers, independent of the spatial layout of the job, and $n - i$ senders, $0 \leq i \leq n$. As shown in Figure 41, the maximum of $\frac{m}{2} * i$ and $\frac{m}{2} * (n - i)$, $0 \leq i \leq n$, cannot be less than $\frac{m*n}{4}$. Thus, the *link_contention* on the vertical link in the middle of the

I/O nodes limits $max\_contention \geq \frac{m*n}{4}$.

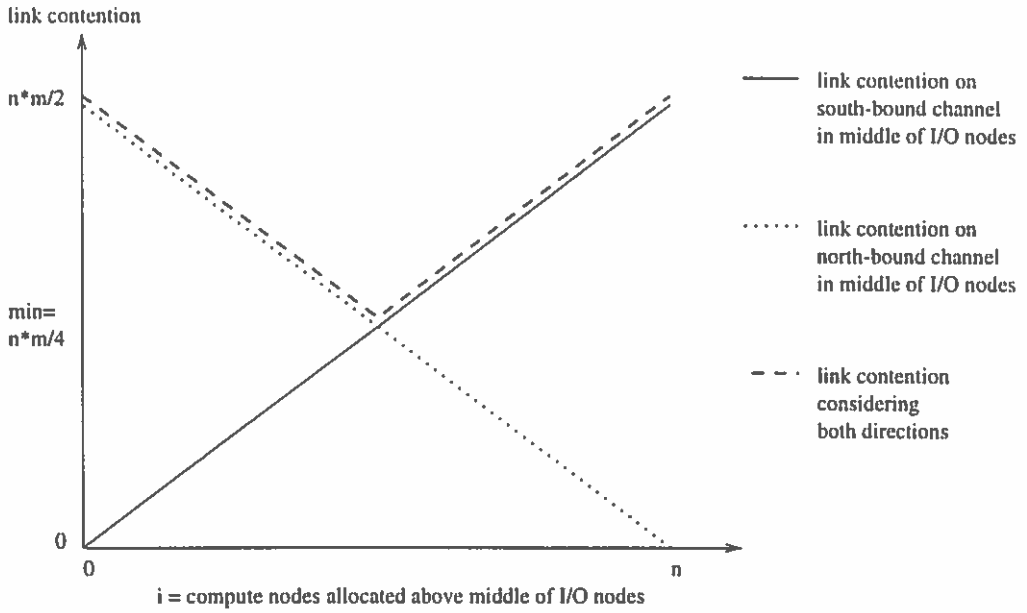


FIGURE 41. Lowest Value for *Link_Contention* on Middle I/O Link

The best case value of $\frac{m*n}{4}$ is achievable, for example when the compute nodes lie parallel in relation to the I/O nodes as shown in Figure 40b. In this spatial layout, the link in the middle of the I/O nodes is the hotspot and $i = \frac{n}{2}$.

Theorem 2: The worst and best case values for network throughput rate are B_l and $4 * B_l$, respectively.

Proof: These formulae are derived directly by substituting the worst and best case values for *max_contention* in the definition of network throughput rate.

The above two theorems confirm that when a complete bi-partite I/O traffic graph is mapped to a mesh architecture with I/O nodes on one side, the network contention occurring on the bottleneck link enforces an upper bound on the network throughput rate. Furthermore, the network throughput rates are independent of the number of I/O nodes.

The Limitations of Adding I/O Nodes

The next theorem describes the limitations of adding I/O nodes to the system configuration. Recall that $f = \frac{B_l}{B_0}$.

Theorem 3: In the best case, at most $4f$ I/O nodes can be used to increase realizable I/O throughput. In the worst case, at most f I/O nodes can be used to increase realizable I/O throughput.

Proof: Theorem 2 tells us that in the best case, $B_{NET} = 4 * B_l$. Thus, $m > 4f \Rightarrow m * B_0 > 4 * B_l \Rightarrow B_{ION} > B_{NET}$. Thus, when m, the number of I/O nodes, is greater than $4f$, realizable I/O throughput is limited to the network throughput rate which is independent of the number of I/O nodes.

Similarly, Theorem 2 tells us that in the worst case, $B_{NET} = B_l$. Thus, $m > f \Rightarrow m * B_0 > B_l \Rightarrow B_{ION} > B_{NET}$. Thus, when m the number of I/O nodes is greater than f, realizable I/O throughput is always limited to the network throughput rate which is independent of the number of I/O nodes.

Below these cutoff points ($m > 4f$ or $m > f$, respectively), realizable I/O throughput depends on $B_{ION} = m * B_0$ which grows with m.

For a mesh architecture with I/O nodes on one side, this theorem gives a much tighter upper bound on the number of I/O nodes than Feitelson et. al. [20], who did not take network contention into account and thus gave an upper bound of $m \leq n * f$.

We next analyze the network contention levels and network throughput rates achieved by our five spatial layout schemes which lie between the theoretical best and worst cases extremes.

Impact of Spatial Layout

Our static analysis shows that network contention levels and realizable I/O throughput are sensitive to the spatial layout of the compute nodes, differing by a factor of four from the best spatial layout to the worst.

Figure 42 shows the hotspot link for each of the five spatial layout schemes for both read and write I/O traffic. Table 5 gives formulas and ranking orders for *max_contention* on the bottleneck link for the case where $m = n = k$, i.e. number of compute nodes equals number of I/O nodes equals one side of the mesh. In general, ranking of spatial layout schemes, from best to worst, matches that shown in Table 5 for read and for write, respectively, across all possible values for m, n, and k. Table 6 gives formulas for network throughput. If the I/O nodes are placed vertically, write traffic is more critical. Parallel and diagonal layouts perform best, followed by block center, block corner and orthogonal. Although diagonal layouts perform well for I/O-intensive jobs, they are highly dispersed and thus perform poorly for communication-intensive workloads, as discussed in Chapter IV.

Simulations

Simulation Model

In order to further quantify the effects of spatial layout on parallel I/O performance, we simulated the realizable I/O throughput rate under different spatial layout schemes. We also varied architectural parameters in order to study the scalability of parallel I/O performance as a function of the number of I/O nodes, the number of compute nodes, and f, the ratio of throughput rate per link to throughput rate per I/O node.

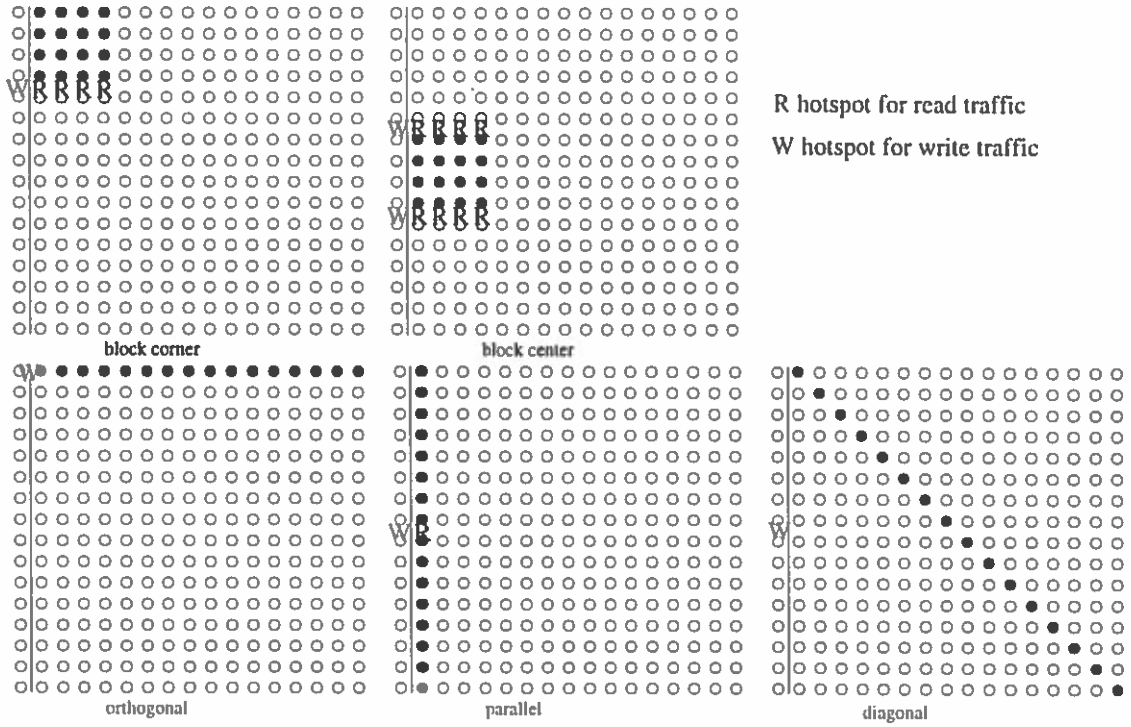


FIGURE 42. Location of Hotspots

TABLE 5. *Max_Contention* and Spatial Layout ($k = n = m$)

Layout	write		read	
	max_contention	ranking	*max_contention*	ranking
diagonal	$\lceil \frac{n}{2} \rceil \lceil \frac{n}{2} \rceil$	1	n	1
parallel	$\lceil \frac{n}{2} \rceil \lceil \frac{n}{2} \rceil$	1	$\lceil \frac{n}{2} \rceil \lceil \frac{n}{2} \rceil$	4
block center	$\frac{n(n-\sqrt{n})}{2}$	2	$\frac{\sqrt{n}(n-\sqrt{n})}{2}$	2
block corner	$n(n-\sqrt{n})$	3	$\sqrt{n}(n-\sqrt{n})$	3
orthogonal	n^2	4	n	1

TABLE 6. Network Throughput and Spatial Layout ($k = n = m$)

Layout	write network throughput	ranking	read network throughput	ranking
diagonal	$B_l * 4$	1	$B_l * n$	1
parallel	$B_l * 4$	1	$B_l * 4$	4
block center	$B_l * 2 * \sqrt{n}/(\sqrt{n}-1)$	2	$B_l * 2 * n/(\sqrt{n}-1)$	2
block corner	$B_l * \sqrt{n}/(\sqrt{n}-1)$	3	$B_l * n/(\sqrt{n}-1)$	3
orthogonal	B_l	4	$B_l * n$	1

More precisely, in these experiments the simulator models the following:

1. mesh architecture of size $k \times k$, $k = 16$ and 32; these are sizes typical of the Intel Paragon and TFLOPS, respectively.

2. m I/O nodes located in a vertical column on the west border of the mesh, $2 \leq m \leq k$;

3. packet level XY-wormhole-routing with one virtual channel;

4. sustainable throughput rates per I/O node of B_0 and sustainable throughputs rates per network link of B_l, $1 \leq f = \frac{B_l}{B_0} \leq 20$;

5. n compute nodes, $2 \leq n \leq k^2$, whose I/O traffic pattern involves exchange of data with every I/O node; thus, each job sends data on $m * n$ different sender-receiver pairs; both read- and write-dominant I/O traffic is modeled;

6. five spatial layout patterns as described above: block corner, block center, parallel, orthogonal, and diagonal (see Figure 39).

The *I/O node throughput rate* is the rate at which all I/O nodes combined can read data from or write data to disks. The interconnection network between nodes is

ignored. The I/O node throughput rate is the aggregate of all throughput rates per I/O node, B_0, and equals $m * B_0$.

The *network throughput rate* is the aggregate data transfer rate through the interconnection network and is measured by the simulator which models packet level, wormhole, XY-routing.

Impact of Spatial Layout

We run extensive simulation experiments to analyze parallel I/O performance as a function of number of compute nodes, number of I/O nodes, f (the ratio of throughput per network link to throughput per I/O node), and spatial layout. Figures 43 and 44 show representative graphs for machines having a 32 x 32 compute mesh and a 16 x 16 compute mesh, respectively, for the case $B_0 \approx 64$ MB/sec per I/O node and $B_l \approx 380$ MB/sec per network link. These examples were selected because the parameters correspond to those of the actual TFLOPS machine. However, our results were consistent across the full range of simulation parameters.

Throughput rates are plotted as functions of jobsize (number of compute nodes n) and number of I/O nodes (m). In Figure 43, the number of compute nodes allocated ranges from 4 to 1024 and the numbers of I/O nodes ranges from 8 to 32. Each of the five graphs represents one of the five spatial layout patterns. In all graphs, the white surface shows the I/O node throughput rate and the shaded surface shows the network throughput rate. The *realizable I/O throughput rate* is the minimum of the I/O node throughput rate and the network throughput rate, depending on whether the I/O nodes or the network are the limiting factor in handling the I/O traffic.

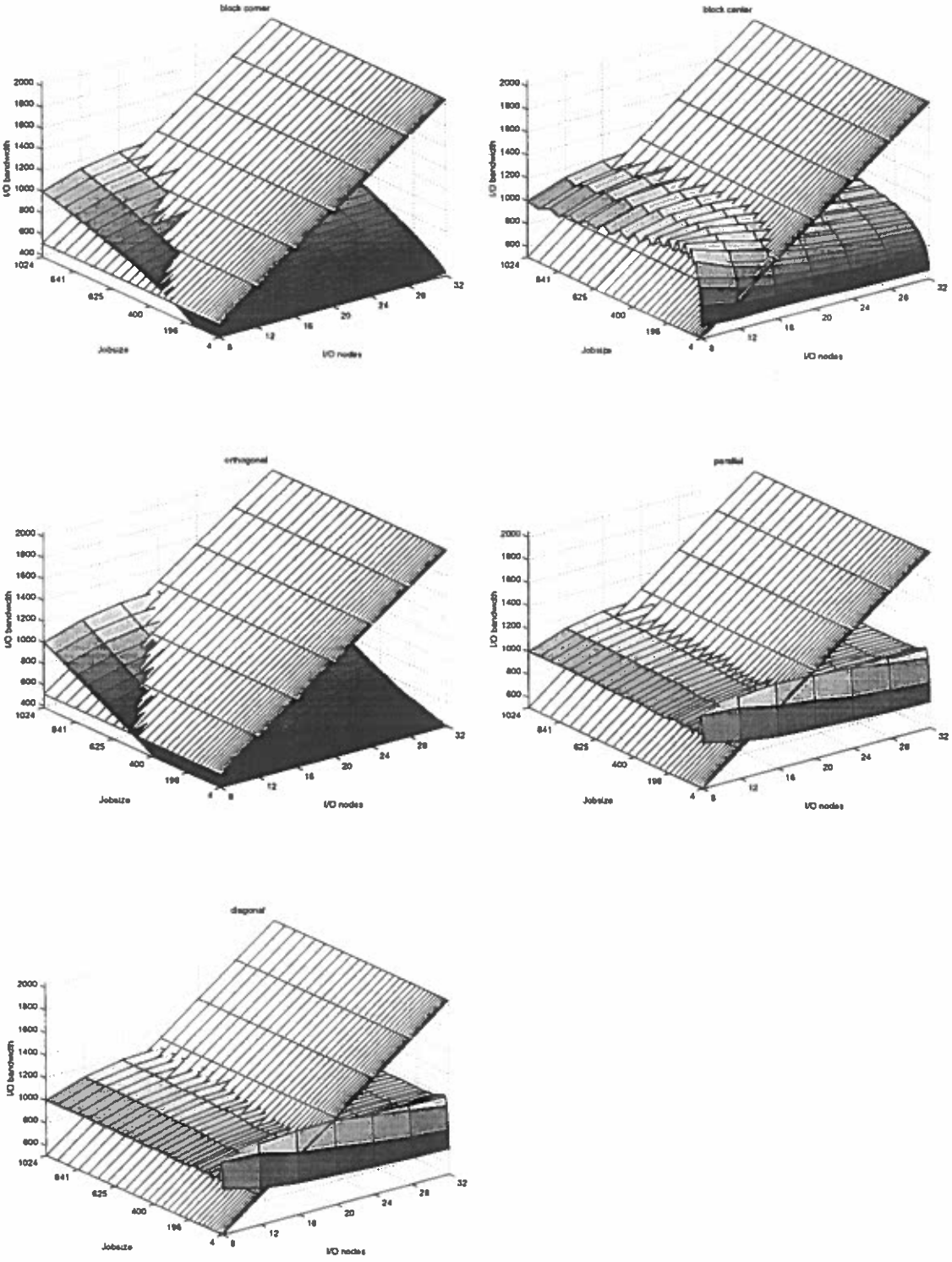


FIGURE 43. I/O Throughput (32×32 Mesh, Write Traffic, $f = 6$)

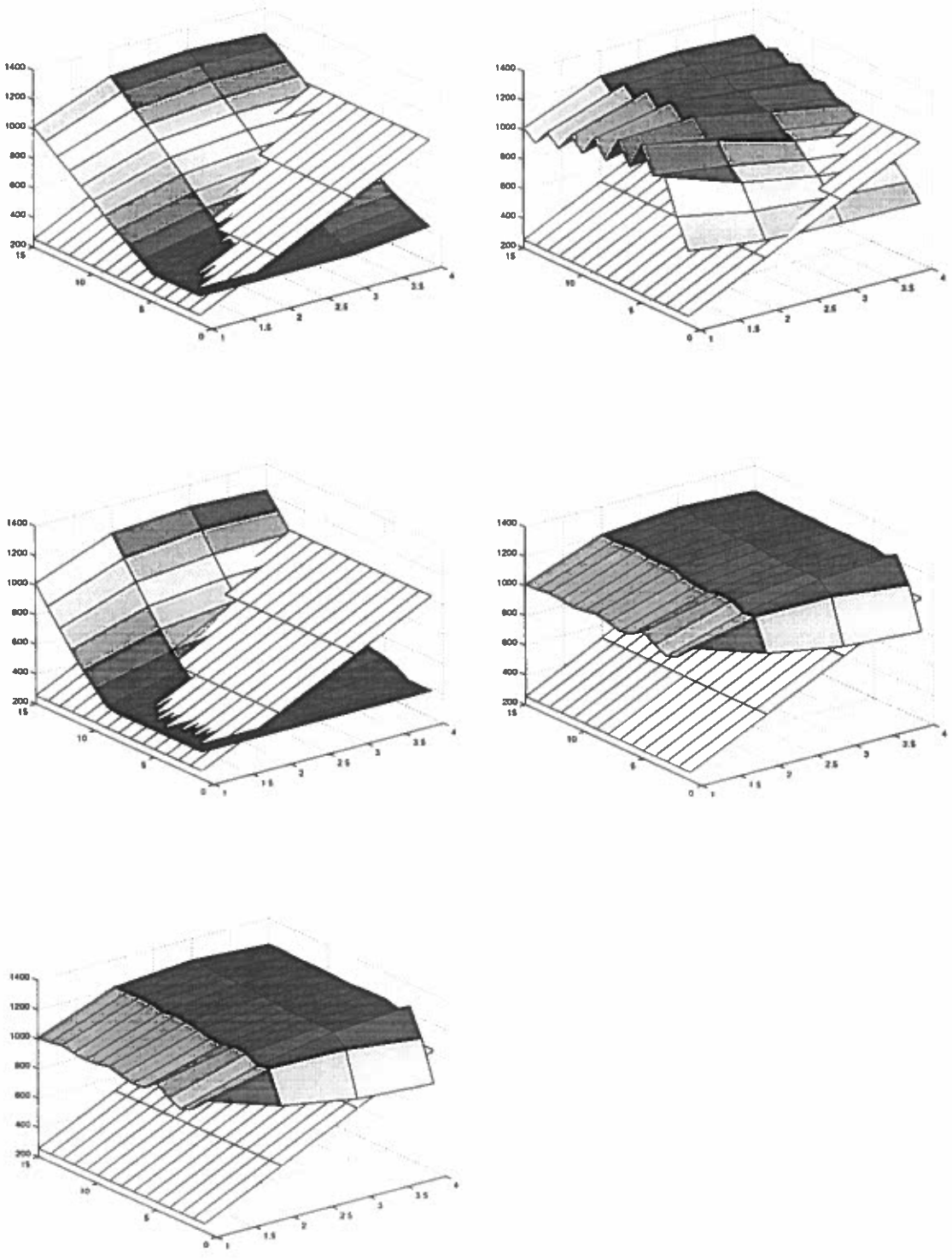


FIGURE 44. I/O Throughput (16×16 Mesh, Write Traffic, $f = 6$)

The key observations to be made from these graphs and Figure 38 are the following:

1. If one looks at throughput versus the number of I/O nodes, it is clear that at some point, adding I/O nodes is useless since the network limits the realizable I/O throughput (i.e., the shaded surface falls below the white surface).

2. The choice of spatial layout affects the degree to which the network limits realizable I/O throughput, and thus affects the cutoff point. For write traffic, the best spatial layout pattern is diagonal or parallel, followed by block center, block corner, and orthogonal. The ranking of spatial layouts for both simulations and analytic modeling from best to worst is consistent over all parameter ranges for m, n, k, and f. The ranking order for read I/O traffic is different than the ranking for write traffic, but is also consistent over all parameter ranges for m, n, k, and f. (The different rankings for read and write traffic are due to XY-routing.)

The impact of this phenomenon is brought home by looking at the specifics of the TFLOPS machine mentioned earlier. Garg et. al. [24] measured I/O throughput of the TFLOPS ASCI/Red where up to 9 I/O nodes were configured into one PFS file system, i.e. every compute node exchanged data with every I/O node. For a big job that utilized most of the machine, realizable I/O throughput scaled with the number of I/O nodes, and nine I/O nodes achieved over 0.5 GB/s throughput. Is it therefore safe to conclude that doubling the number of I/O nodes from 9 to 18 will double the I/O throughput to 1 GB/s? Our work indicate that it depends on the spatial layout scheme whether 18 I/O nodes can be used

productively and thus whether an I/O throughput of one GB/s is achievable (see Figure 38).

We summarize our results in Figure 45 which shows throughput as a function of the number of I/O nodes. In this figure, the diagonal line gives the I/O node throughput rate (ignoring the limitations of the network). The two horizontal lines show the theoretical best and worst case values of network throughput, $4 * B_l$ and B_l, respectively. The other five lines show the network throughput rate for the five spatial layouts.

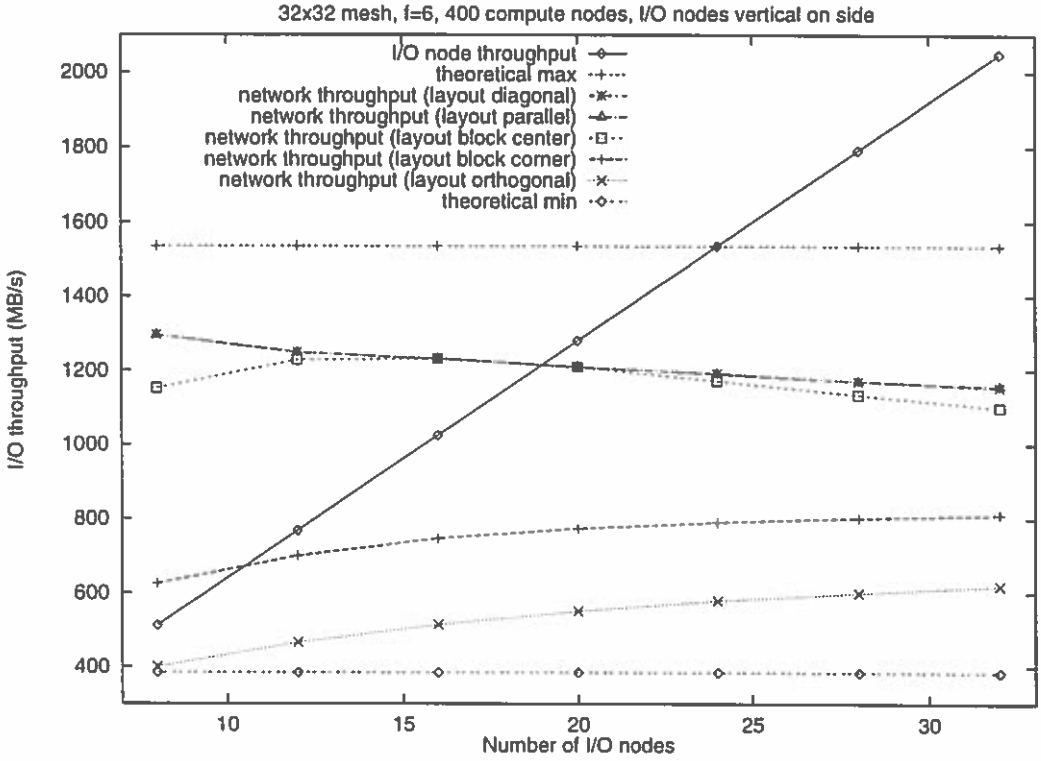


FIGURE 45. I/O Throughput and Spatial Layout (32×32 Mesh)

From this graph, we see that depending on the choice of spatial layout scheme, we can achieve network throughput rates more or less close to the theoretical best case. We can also see that given, a spatial layout scheme, the point where the line

for I/O node throughput intersects that for network throughput is the point at which addition of I/O nodes has no further effect.

Conclusions

This chapter focuses on analyzing traffic hotspots due to data transfer between compute nodes and I/O nodes. We assume mesh architectures, XY-routing, complete bipartite I/O traffic graphs and I/O nodes configured on one side of the mesh. Our analytic modeling together with extensive simulations lead us to conclude the following:

1. *Network contention* can limit parallel I/O performance. On a 2D mesh with I/O nodes on one side, realizable I/O throughput is – in the best case – limited to 4 times the effective throughput per network link, B_l. Thus, adding I/O nodes (of effective throughput rate B_0) does not pay beyond $4 * B_l/B_0$. This is a much tighter upper bound on the number of I/O nodes than $n * B_l/B_0$ [20] which did not take network contention into account.

2. *Spatial layout of jobs* in relation to the I/O nodes affects network contention and thus parallel I/O performance. If the I/O nodes are on one side of the mesh, tuning the spatial layout of jobs makes a difference of up to a factor of 4. If network throughput is the bottleneck, spatial layout must be taken into account. Processor allocation strategies that are sensitive to parallel I/O traffic should be able to improve parallel I/O performance significantly.

3. Depending on the layout of the I/O nodes in the mesh, one direction of parallel I/O traffic performs much worse than the other direction. If the I/O

nodes are placed vertically, write traffic (to the I/O nodes) is the bottleneck. If they are placed horizontally, read traffic is the bottleneck. These effects are a result of XY-routing.

CHAPTER VII

MINIMIZING I/O-BASED NETWORK CONTENTION

Motivation

In this chapter, we present several new processor allocation strategies that minimize I/O-based network contention, thereby improving I/O throughput and overall system performance. First, we present a strategy called *PLAS* that is optimized exclusively for parallel I/O-traffic. Our approach in tackling the traffic hotspot problem (that we analyzed in Chapter VI) is to concentrate on minimizing contention on the "middle" I/O link and to strive for *balance* when allocating jobs to compute nodes.

Because many scientific jobs have both I/O and communication needs, we then devise a strategy, *MC Elongated*, that is optimized for both parallel I/O and intra-job communication. Our approach is to minimize contention by striving for both balance and allocation compactness.

We test the performance of our new allocation strategies with simulations driven by synthetic workloads and by real workload traces. We report on overall performance and its sensitivity to system load and to different ratios of I/O traffic vs. communication traffic.

Our results show that with respect to system throughput and job response time, our new strategies consistently outperform known allocation strategies that are in use on commercial and research machines. Average response times of jobs

improved by up to a factor of 4.5.

Parallel I/O-Sensitive Allocation Strategies

Whereas previous non-contiguous allocation strategies have essentially ignored I/O-based network contention, our new strategies are sensitive to the impact of spatial layout on parallel I/O throughput.

Balance Factor

Recall from Chapter VI that *link contention* of link l is the number of distinct sender-receiver pairs sharing network link l. A link with maximal link contention is called the *hotspot* and its value of link contention is called *max_contention*.

Under the assumption of I/O write traffic and I/O nodes placed vertically on one side of the mesh, we showed that achieving the lowest value for *max_contention* requires that exactly one half of all allocated compute nodes are in rows above the middle I/O link. This spatial layout is balanced with respect to the middle I/O link.

The effect of "unbalanced" spatial layout on network contention is shown in Figure 46. In contrast to the balanced layout on the left, the layout on the right has 4 compute nodes allocated in the upper half and only 2 compute nodes allocated in the lower half of the system. Thus, assuming all compute nodes send data to all I/O nodes, the number of sender-receiver pairs using the middle I/O link is $2 * 2$ upwards and $4 * 2$ downwards in the unbalanced, instead of $3 * 2$ upwards and $3 * 2$ downwards in the balanced case. Each direction has its own physical channel. For the layout on the right, the I/O traffic distribution is more uneven.

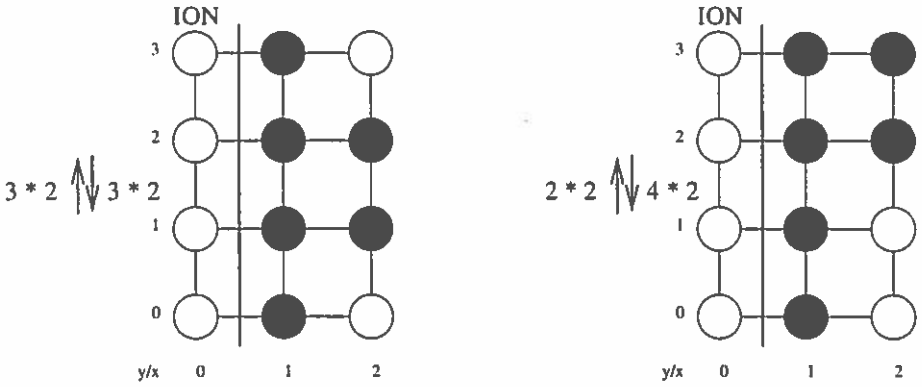


FIGURE 46. Effect of Balanced and Unbalanced Spatial Layout on Network Contention

This motivates us to search for a metric that captures the degree of balance in a given allocation, as a way to minimize *max_contention*. In addition, we want the metric to be efficient to compute. Our metric *balance factor* is defined in the following way:

Definition: Given a job A and its allocation to a set of compute nodes N, *above_middle* of A equals the subset of N which reside in rows above the middle I/O link. The *middle I/O link* is the link between I/O node $\frac{m}{2}-1$ and $\frac{m}{2}$. Similarly, *below_middle* of A equals the subset of N which reside in rows below the middle I/O link. The *per job balance factor* of A is defined as the difference between *above_middle* of A and *below_middle* of A.

Definition: Given a *snapshot* of the mesh at time t, there may be j_t jobs currently allocated. The *system balance factor* at time t is defined as the sum of the per job balance factors of all j_t jobs. It is equivalent to $\sum above\_middle - \sum below\_middle$ considering all jobs currently allocated.

Definition: Given a workload and an allocation strategy, *average balance factor* is defined as the average of the absolute values of the system balance factor

taken at times jobs are being allocated or deallocated.

Definition: A per job balance factor or a system balance factor is *perfect* if its value is 0, 1 or -1. An average balance factor is perfect if its value is 0.

Figure 47 shows an example of five jobs. Job A has size 2, and it is assigned one node above and one node below the middle I/O link. Thus, its balance factor is 0. Of the other jobs, jobs D and E have odd sizes and their balance factors are +1 and -1, respectively. The balance factor of the current snapshot is zero: 13 allocated compute nodes reside in rows above the middle I/O link and 13 allocated compute nodes reside in rows below the middle I/O link.

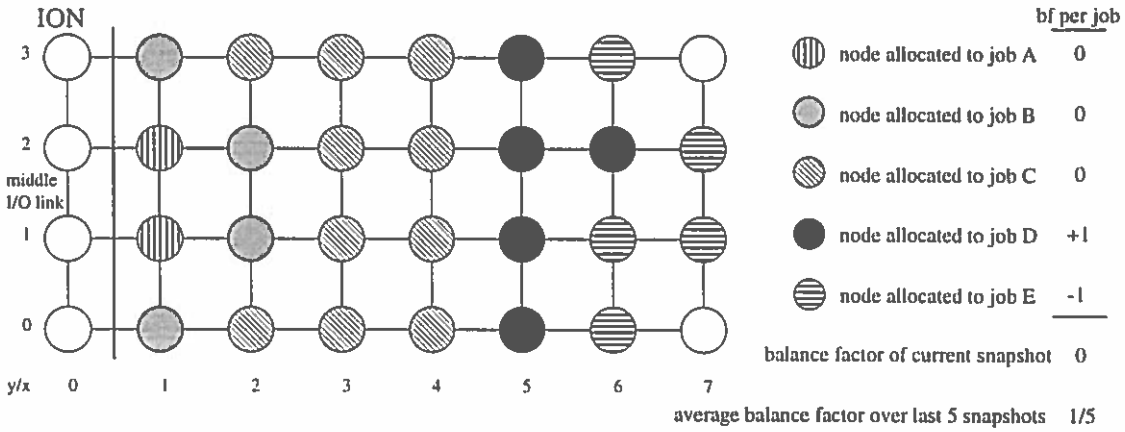


FIGURE 47. Balance Factors in a Snapshot of Five Allocated Jobs

In an empty system, the balance factor of the snapshot is zero. As the five jobs arrived, the balance factors of the snapshots were the following: 0 after job A was allocated, 0 after job B was allocated, 0 after job C was allocated, +1 after job D was allocated and 0 after job E was allocated. The average balance factor over these 5 arrival events is 1/5. This is the best possible considering that we have two jobs of odd size.

Corollary to Theorem VI.1: Given a set of jobs that are in the system at the

same time, an allocation that achieves minimal *max_contention* under complete bipartite I/O write traffic has perfect system balance factor.

This follows directly from the definition of balance factor and from the proof of Theorem 1 in Chapter VI. Thus, perfect balance is a necessary condition for minimal *max_contention*.

Allocation algorithms that are designed to strive for perfect balance are able to quickly eliminate the worst possible spatial layouts with respect to *max_contention*. This can best be explained with an example. Given a 4×4 compute mesh with 4 I/O nodes on one side, Figure 48 shows how allocating a job of size 4 in an empty system affects *link_contention*. In this matrix M, the entries in row l_i represent possible *link_contention* values on the link l_i. A single entry in this matrix specifies the *link_contention* on link l_i for the specified value of *senders_above*, the number of compute nodes allocated above link l_i, according to the spatial layout. Thus, any specific allocation of 4 compute nodes corresponds to the selection of one entry from each column under the constraint that *senders_above* of l_{i+1} has to be at least as big as *senders_above* of l_i. Striving for perfect balance on the middle I/O link, we allocate our job of size 4 such that *senders_above* of the middle I/O link (l_2) is 2 by choosing column 2 in row l_2. This choice eliminates columns 3 and 4 in rows l_1 and l_2, and columns 0 and 1 in rows l_2 and l_3 (see Figure 48) . Thus, we eliminated spatial layouts that could have caused the worst *link_contention* values of 6, 8, 9, and 12. Figure 50 shows a bigger example, generated by the code in Figure 49. A perfectly balanced allocation selects column 7 in row l_8, thereby eliminating allocations with the highest levels of *link_contention*.

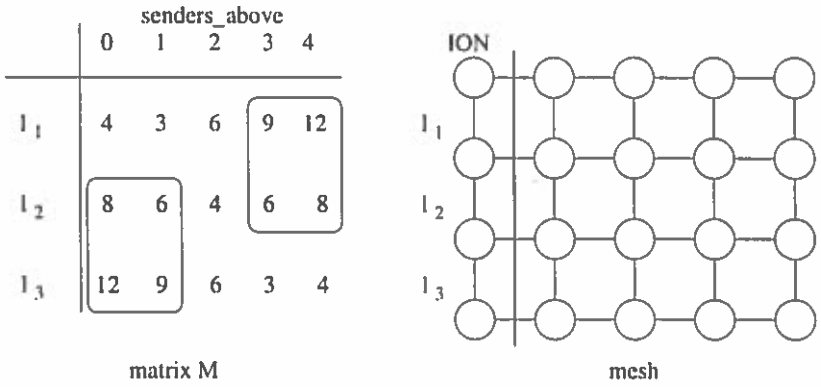


FIGURE 48. *Link_Contention* When Allocating a Job of Size 4 in an Empty 4×4 Compute Mesh

```
for (link = 1; link < sidelength; link++) {
   receivers_above = link - (sidelength-num_ion) / 2;
   receivers_above = max(receivers_above, 0);
   receivers_above = min(receivers_above, num_ion);
   for (senders_above = 0; senders_above <= jobsize; senders_above++) {
      link_contention = max(senders_above*(num_ion-receivers_above),
                            (jobsize-senders_above)*receivers_above);
   }
}
```

FIGURE 49. Algorithm to Compute the Matrix of Figure 48 and Figure 50

```
Sidelength of the mesh: 16
Number I/O nodes      : 16
Size of the job(s)    : 14
 14  15  30  45  60  75  90 105 120 135 150 165 180 195 210  min= 14
 28  26  28  42  56  70  84  98 112 126 140 154 168 182 196  min= 26
 42  39  36  39  52  65  78  91 104 117 130 143 156 169 182  min= 36
 56  52  48  44  48  60  72  84  96 108 120 132 144 156 168  min= 44
 70  65  60  55  50  55  66  77  88  99 110 121 132 143 154  min= 50
 84  78  72  66  60  54  60  70  80  90 100 110 120 130 140  min= 54
 98  91  84  77  70  63  56  63  72  81  90  99 108 117 126  min= 56
112 104  96  88  80  72  64  56  64  72  80  88  96 104 112  min= 56
126 117 108  99  90  81  72  63  56  63  70  77  84  91  98  min= 56
140 130 120 110 100  90  80  70  60  54  60  66  72  78  84  min= 54
154 143 132 121 110  99  88  77  66  55  50  55  60  65  70  min= 50
168 156 144 132 120 108  96  84  72  60  48  44  48  52  56  min= 44
182 169 156 143 130 117 104  91  78  65  52  39  36  39  42  min= 36
196 182 168 154 140 126 112  98  84  70  56  42  28  26  28  min= 26
210 195 180 165 150 135 120 105  90  75  60  45  30  15  14  min= 14
```

FIGURE 50. *Link_Contention* When Allocating a Job of Size 14 in an Empty 16×16 Compute Mesh

PLAS

PLAS – Parallel Layout Allocation Strategy – is a heuristic that aims at perfect balance in order to minimize contention due to I/O traffic. When it is asked to allocate a new job, it scans for idle compute nodes in a predefined scanning order (see Figure 51), always starting from the same "middle" point. Applying the results from Chapter VI that show that placement of compute nodes parallel to the I/O nodes yields best overall performance, we designed PLAS to scan parallel to the I/O nodes, one column after another. In each column, PLAS scans "middle-out". In the example of Figure 51, a job requesting six compute nodes would get the compute nodes labeled 1 through 6 if these are currently idle. Until it has found enough idle nodes to fulfill the request, PLAS keeps on searching according to the scanning order. The spatial layouts in Figure 47 were produced by PLAS.

Pseudocode is shown in Figure 52.

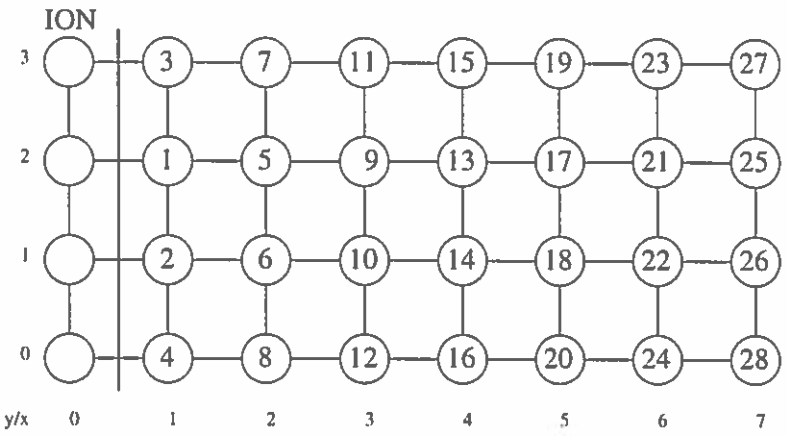


FIGURE 51. Scanning Order of PLAS (Parallel Layout Allocation Strategy)

```
plas_allocate(jobsize){
  initialize column=1, middle=k/2, offset=0 and sign=+1
  while (jobsize > 0) {
    node_x = column; node_y = middle + sign * offset
    if node idle then allocate node; jobsize--
    else {
      flip sign
      every second time increase offset
      every k times increase column and reset offset
  }
}
```

FIGURE 52. Pseudocode of PLAS

The characteristics of PLAS are the following:

Property: Starting with an empty system and allowing no job departures, PLAS achieves perfect per job balance factors and perfect system balance factors.

Proof: The scanning order makes sure that nodes to be assigned are taken in alternation, one from above and one from below the middle I/O link. Thus both the per job and system balance factors are always 0 if the total number of nodes

allocated is even or +1 if the total number of nodes allocated is odd. Because in the scanning order, the next job picks up where the previous job stopped, an even job does not change the system balance factor, whereas an odd job changes the system balance factor from 0 to +1 or from +1 to 0.

In the general case, job departures are allowed. This can lead to system balance factors and per job balance factors that are not perfect. For example, if several jobs with per job balance factor of +1 leave, the system balance factor will not be perfect anymore. If an arriving job fills several of these holes, its per job balance factor will not be perfect either. Theoretically, both per job balance factor and system balance factor can only be bounded by half the size of the compute mesh, the worst possible case.

However, in our simulations (see below) we saw that PLAS quickly converges back to perfect system balance. Arriving jobs fill the holes because PLAS always scans in the same order, starting from the same point. When all holes are filled, the system is balanced again.

Performance Evaluation

We conducted extensive simulation experiments to compare PLAS to previous non-contiguous allocation strategies under heavy I/O traffic. As before, we use our ProcSimity simulator [57]. We model wormhole switching, minimal dimension-ordered XY routing and mesh topologies with two uni-directional links between adjacent nodes. However, this time we assume a configuration with I/O nodes on one side of the mesh, and we model each job as I/O intensive (complete bipartite I/O write traffic).

We compare PLAS to three previous non-contiguous allocation strategies described in Chapter III, two of which are in use on commercial and research machines. These strategies were not designed to be sensitive to parallel I/O traffic.

1. *MBS* [35] is a block-based strategy that typically allocates several blocks of sizes $2^i \times 2^i$. MBS is similar to the M2DB strategy[53] which has been used at the San Diego Supercomputing Center.

2. *Paging* [35] scans for idle compute nodes in row-major order, and thus typically allocates several rows of compute nodes. Paging was shown to have very good performance for computation or communication-intensive workloads [35] and it has been used at NASA Ames Research Center.

3. Under *Random*, a request for n compute nodes is satisfied with n randomly selected compute nodes.

Our experiments are designed to test whether I/O sensitive strategies outperform previous strategies that do not take I/O-based network contention into account. To allow for both a realistic evaluation and for easy comparison with previous experiments, our performance evaluation again includes the use of a synthetic workload and a real workload trace. As summarized in Table 7, Workload I is a synthetic workload, a stream of 1000 jobs whose sizes and interarrival times are exponentially distributed. Workload II is a trace-derived workload. a stream of 6087 real production jobs recorded over a three months period from October 1 to December 31, 1996, from the Intel Paragon at the San Diego Supercomputer Center. The traced job stream is taken only from the 352 node NQS partition [56] of the machine, through which all batch jobs were scheduled. The traces had the

TABLE 7. Characterization of the Synthetic Workload and the Real Workload Trace

	jobsize	interarrival time	number of jobs
I: synthetic	exponential mean = $4 * 4$	exponential mean varies	1000
II: real (SDSC)	mean = 14.5 c.v. = 1.5	mean varies c.v. = 3.7	6087

statistical characteristics shown in Table 7.

To challenge these allocation strategies, we multiply job arrival times by constant factors c. As c decreases, simulated interarrival times decrease, resulting in increased system load. As c increases, simulated interarrival times increase and system load decreases. The simulated interarrival time is chosen such that average system utilization ranges from about 10% to about 90%.

Since our focus is on traffic hotspots due to parallel I/O traffic, we first model a heavy I/O load, no computation and no communication. For both Workload I and Workload II, job service time depends on the amount of I/O traffic generated and the amount of network contention encountered. Each job transfers data according to a complete bipartite I/O pattern, i.e. each compute node allocated to the job sends a personalized message to every I/O node. The number of messages sent is correlated to jobsize because we have each job execute the same number of iterations of the complete bipartite I/O pattern before it leaves the system.

In the results reported, we model a mesh of 16×22 compute nodes, matching the batch partition of the Intel Paragon at SDSC. We assume a configuration where 16 I/O nodes are located vertically on the west side of the compute nodes. We assume that compute nodes write to I/O nodes (i.e. write traffic only) since the analysis in Chapter VI showed that the hotspots due to write traffic are much more

severe than the hotspots due to read traffic if I/O nodes are placed vertically.

We simulate the detailed message-passing behavior at the level of packets. For our first experiment, we measure the following performance metrics: average response time, average service time, average max_contention and average balance factor.

Simulation Results

Figure 53 and Figure 54 show average response times for both Workload I and Workload II, respectively. For the SDSC workload, PLAS results in up to a factor of 2.2 speedup in average response time compared to previous non-contiguous allocation strategies that were not parallel I/O sensitive. For the synthetic workload, PLAS results in up to a factor of 4.5 speedup in average response time. Clearly, tuning the spatial layout of compute nodes resulted in impressive performance gains. We are pleased that PLAS outperforms Random, which is expected to perform well because it distributes I/O traffic in a balanced manner. (However, we do know that Random is a bad choice for communication-intensive workloads.)

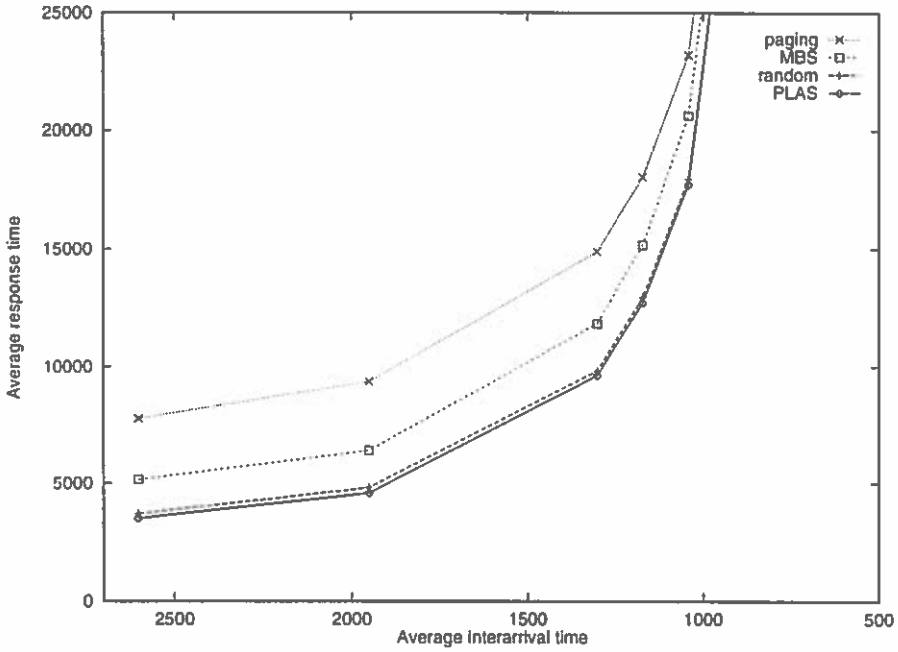


FIGURE 53. Average Response Time (SDSC Workload of 6087 I/O-Intensive Jobs)

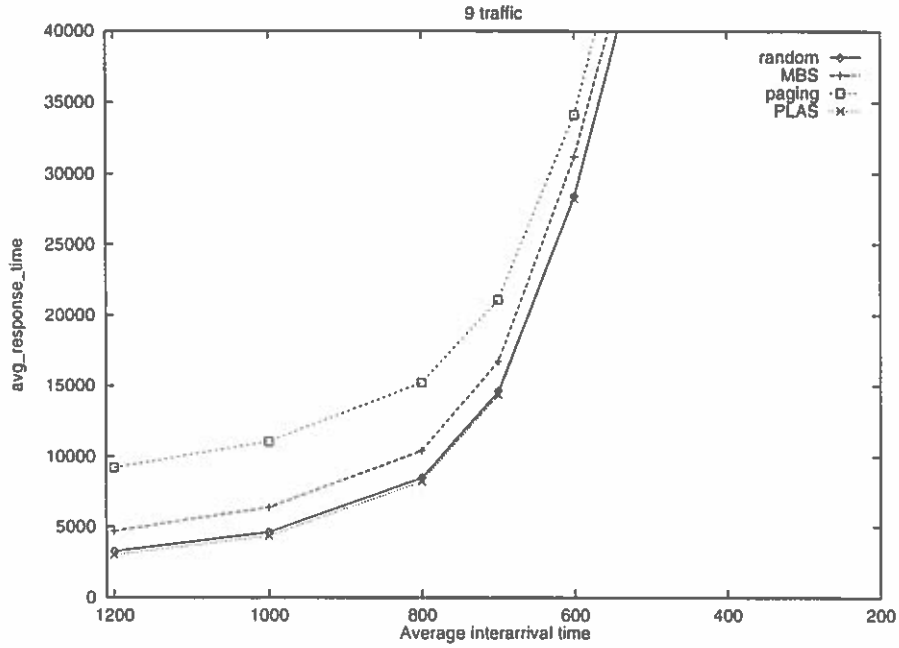


FIGURE 54. Average Response Time (Synthetic Workload of 1000 I/O-Intensive Jobs)

We attribute PLAS's superior performance to its ability to balance the system. Figure 55 shows average balance factor for different allocation strategies over our synthetic workload of 1000 jobs. Our simulations show that PLAS dramatically outperforms the other allocation strategies in achieving balance.

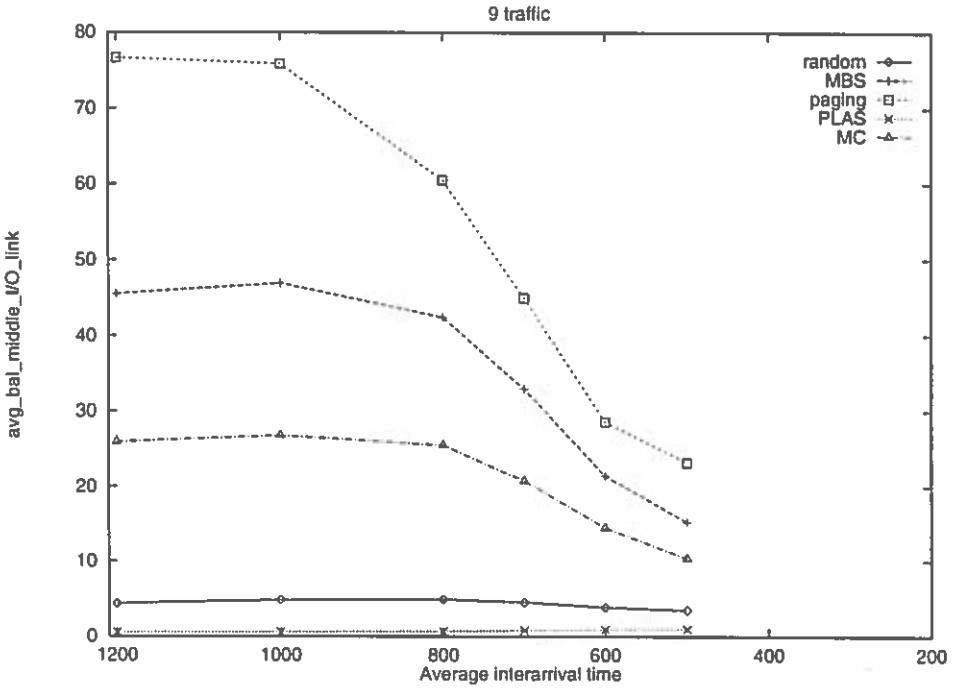


FIGURE 55. Average Balance Factor (Synthetic Workload of 1000 I/O-Intensive Jobs)

Recall that the corollary above indicates that the absolute value of system balance factor must be as close to 0 as possible to achieve minimal max_contention. Our simulations show the average balance factor of PLAS to be strikingly closer to perfect than those of paging, over a range of system loads (average interarrival times from 1600 to 200, resulting in system utilizations between 6% and 85%). The average balance factor of PLAS ranged from 0.44 to 0.90 with an average of 0.64 (increasing as the system load increased), whereas the average balance factor of Paging ranged from from 28.02 to 80.63 with an average of 52.73 (mainly

decreasing as the system load increased). Our simulations also show that striving for perfect balance pays off in max_contention (see Figure 56) and service time (see Figure 57).

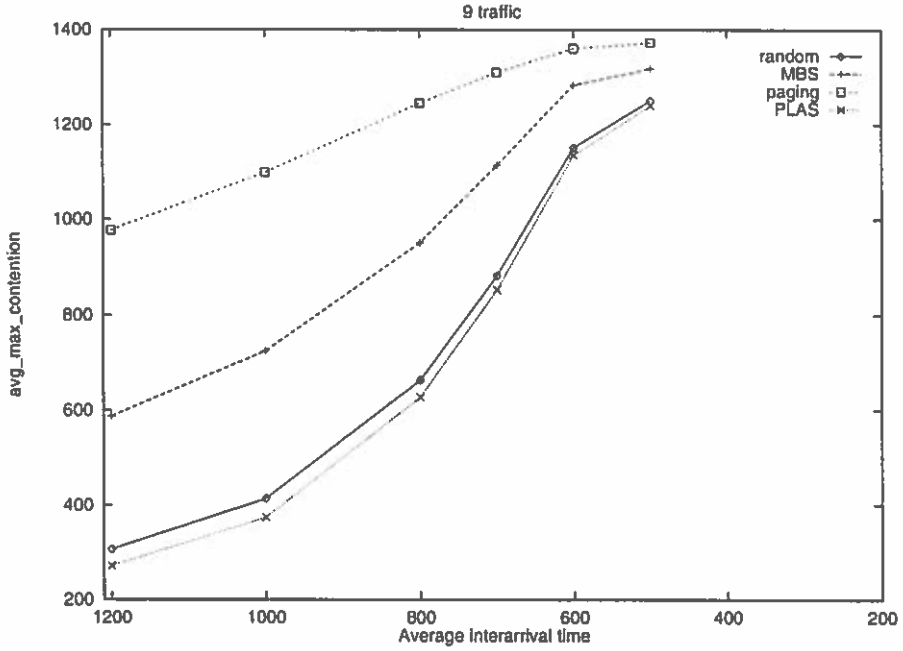


FIGURE 56. Average *Max_Contention* (Synthetic Workload of 1000 I/O-Intensive Jobs)

Table 8 shows the values of average balance factor and *max_contention* for the synthetic workload and average interarrival time of 500. The correlation between average balance factor and *max_contention* is very strong, the Pearson correlation coefficient is 0.988319.

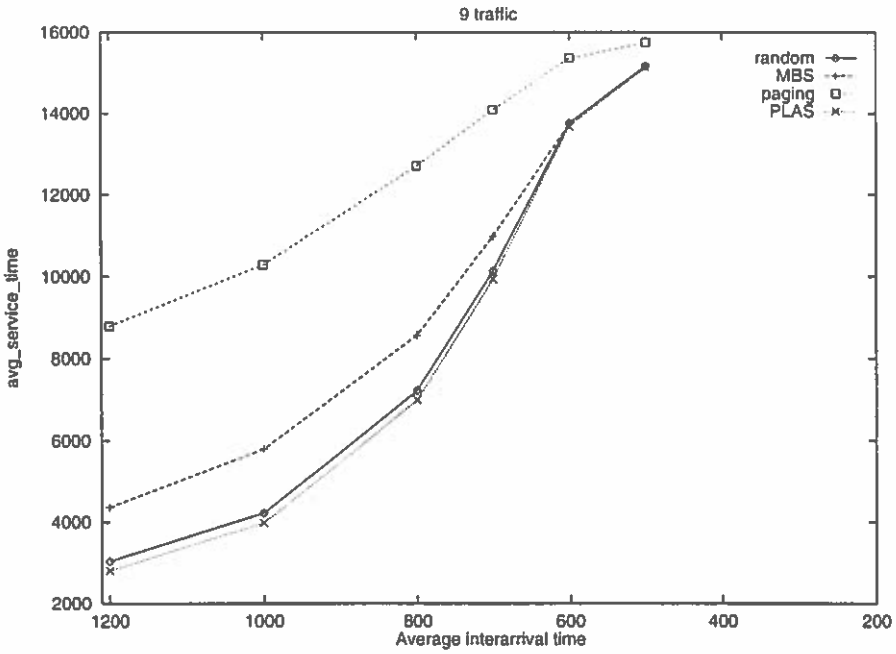


FIGURE 57. Average Service Time (Synthetic Workload of 1000 I/O-Intensive Jobs)

TABLE 8. Balance Factor vs. *Max_Contention*

strategy	avg. balance factor	max_contention
Paging	32.142500	1336.475000
MBS	19.717500	1224.895000
MC	15.430000	1199.340000
MC Elongated	7.005000	1152.980000
Random	4.425000	1140.740000
PLAS	0.687500	1124.877500

Parallel I/O- and Communication-Sensitive Allocation

PLAS, our new strategy described above, alleviates hotspots and thus achieves performance improvements for I/O-intensive workloads. How does PLAS perform on communication-intensive workloads?

Figure 66 shows that on communication-intensive workloads, PLAS does not perform that well in comparison to MC, which is optimized for communication-intensive workloads. Figure 58 gives an explanation of the shortcoming of PLAS. Given the situation in Figure 58a in which the black job is to be allocated nodes, PLAS fills the holes in column 1 and achieves perfect system balance factor, but the allocation is not *compact* (see Figure 58b). When the compute nodes of the job communicate instead of doing I/O, this spatial layout results in high inter-job link contention.

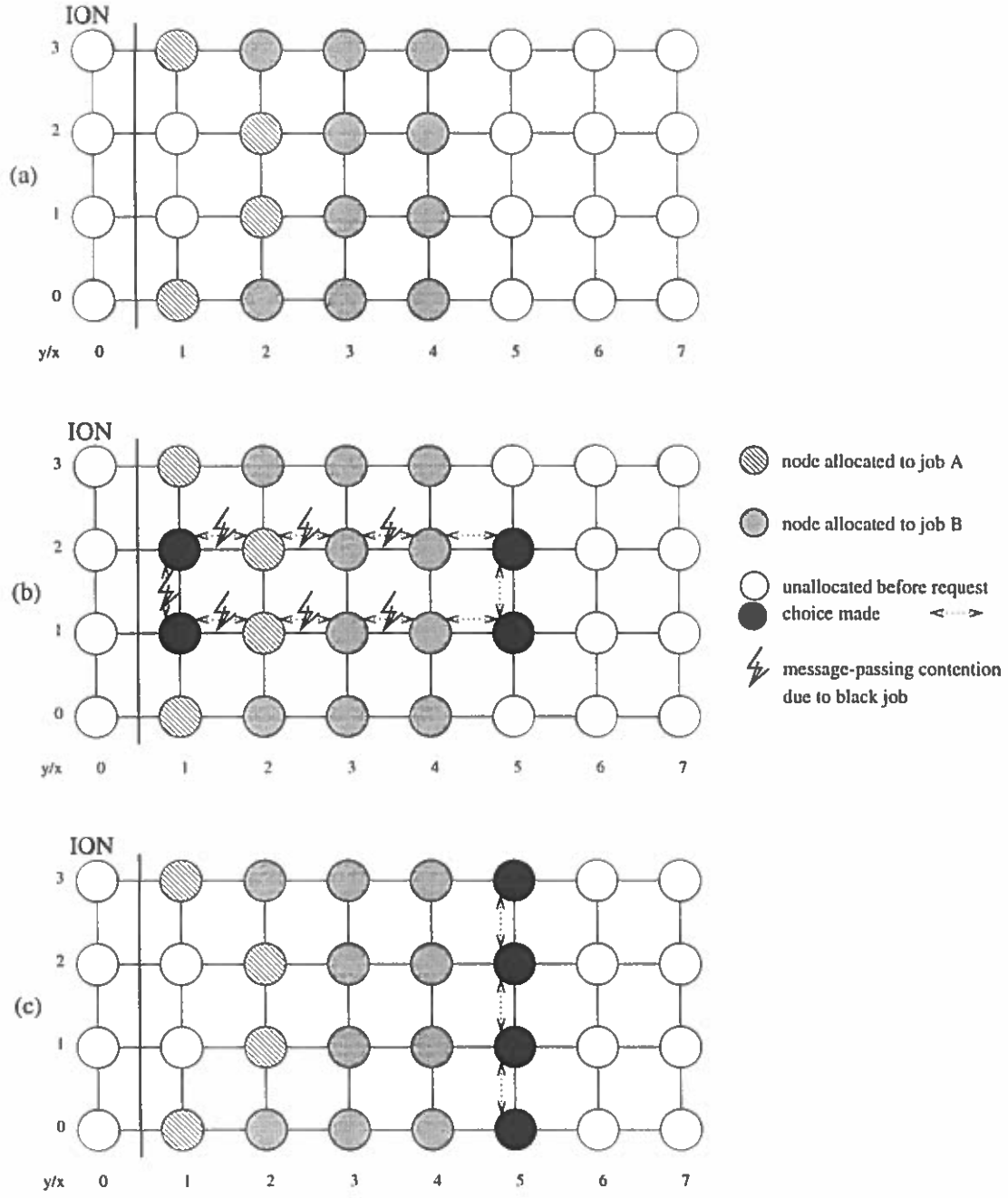


FIGURE 58. Inter-Job Link Contention of PLAS vs. MC Elongated: Original Situation (a), Spatial Layout According to PLAS (b), Spatial Layout According to MC Elongated (c)

Conversely, how does MC, the strategy tuned exclusively for communication traffic (see Chapter V), do on I/O-intensive workloads? Figure 61 shows that on I/O-intensive workloads, MC does not perform that well in comparison to PLAS and Random. The shortcoming of MC is that its allocations are not necessarily *balanced*, as we saw in Figure 55.

These observations motivated us to design a strategy that considers both I/O traffic and intra-job communication traffic.

MC Elongated

MC Elongated is a new processor allocation strategy that is optimized for both parallel I/O- and communication-intensive workloads. MC Elongated strives for allocation compactness and balance at the same time.

MC Elongated is a modification of MC (see Chapter V). As in MC, each idle compute node executes a shell-like scan to find and evaluate a candidate cluster made up of idle nodes in its neighborhood. MC Elongated uses the same cost metric as MC.

In contrast to MC, MC Elongated tries to fill whole columns in order to achieve an allocation with parallel shape and good balance. Comparing the pseudocode of MC Elongated (see Figure 59) to the pseudocode of MC (see Figure 31 in Chapter V) reveals one big difference: $Shell_0$, the nodes that can be added to the candidate cluster for a cost of 0, contains $\lfloor jobsize/k \rfloor$ columns, where k is the height of the compute mesh. Since the candidate cluster with minimal total cost is chosen, this ensures that in most cases a contiguous block of $\lfloor jobsize/k \rfloor$ columns gets allocated. Figure 58c showed an example with $k = jobsize = 4$. This

property results in compact allocations with perfect job balance factor if such a spatial layout is available and if jobsize is divisible by k.

```
mc_allocate(jobsize){
  if number_idle < jobsize then return fail
  for each idle node (i, j)
    cluster = empty list; tcost = 0
    add idle nodes to cluster that are in
      jobsize/k x k rectangle around (i, j) /* shell_0 */
    for each shell s >= 1
      stop if |cluster| = jobsize
      if node idle then add to cluster; tcost += s
  select cluster with minimal tcost
  allocate those nodes, update number_idle and busy array
}
```

FIGURE 59. Pseudocode of MC Elongated for $k \times k$ Mesh

If $shell_0$ does not contain enough idle nodes, recall from Chapter V that to construct a candidate cluster, idle nodes are selected from successive shells, beginning with $shell_0$, until enough idle nodes have been found to satisfy the request. In the original MC algorithm, the order in which idle nodes are selected within each shell (shorter sides first, longer sides second, corners last) was tuned for optimal degree of compactness. However, in MC Elongated each shell adds one more column to the left and one more column to the right; within each column the scanning order is "middle out" (as in PLAS) such that balanced allocations are achieved.[1]

MC Elongated has the following characteristics:

[1] In the results reported, MC Elongated did not use this new order yet; it used the old order according to the MC algorithm instead. This explains the balance factors in Figure 71. If we do use the new order, we expect even better results regarding balance and performance under I/O-intensive workloads.

Property: Like MC, MC Elongated never leaves idle nodes unselected on communication paths between selected nodes.

This is due to the shell-like scanning scheme and the definition of cost. This characteristic leads to compact allocations, lower inter-job link contention and good performance under communication-intensive workloads.

Property: Like MC, MC Elongated is neither conservative nor block-based.

This is in contrast to PLAS, Figure 58b showed that PLAS is conservative in that it first fills the holes before it utilizes columns further to the right. As explained in Chapter V, both conservative and block-based approaches are "eager" and pick up small leftovers before touching bigger idle spaces, in anticipation of saving the bigger space for a bigger job that may be coming later. As explained before, this conservative attitude almost always results in less compact allocations, and in many situations, it does not even help the allocation of future jobs. The conservative approach has no advantage if either the preserved block will be split up anyways by a later job that is not that big or if the preserved block will not be used anyways because either no big job arrives before other jobs leave or the next big job is too big for the preserved block anyways. The approach of MC Elongated is more along the line of *cape diem*, seize the day.

However, while MC always allocates a contiguous cluster if one exists, MC Elongated sacrifices this property in order to achieve better balance. Figure 60 shows an example where MC would allocate a contiguous cluster (achieving the highest possible degree of compactness), whereas MC Elongated allocates a cluster that is less compact, possibly causing inter-job link contention.

Property: Like PLAS, starting with an empty system and allowing no job

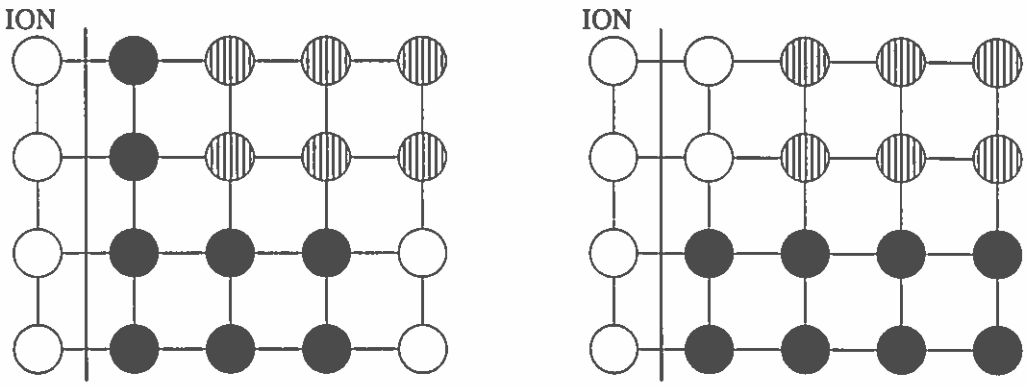


FIGURE 60. How MC Elongated (Left) and MC (Right) Allocate a Job of Size 8

departures, MC Elongated achieves perfect per job balance factors and perfect system balance factors.

However, since MC Elongated concentrates on compact allocations instead of on filling holes, MC Elongated does not re-balance the system as quickly as MC, in general situations including job departures.

Performance Evaluation

We conducted experiments to compare MC Elongated to the other non-contiguous allocation algorithms. We varied the percentage of I/O traffic vs. communication traffic from 100% I/O traffic to 0% I/O traffic in increments of 20% (and 25%). As always, we are interested in average response time, especially average service time. Figure 61 through Figure 66 show average service time for the synthetic workload under different ratios of I/O traffic vs. communication traffic. Similarly, Figure 67 through Figure 70 show average service time for the SDSC workload.

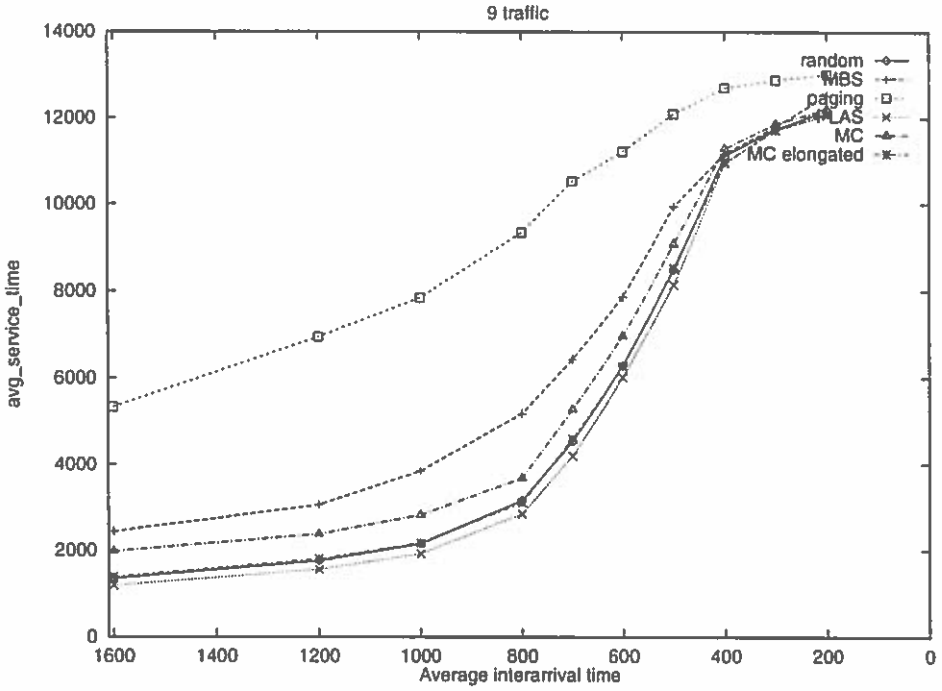


FIGURE 61. 100% I/O Traffic (Synthetic Workload)

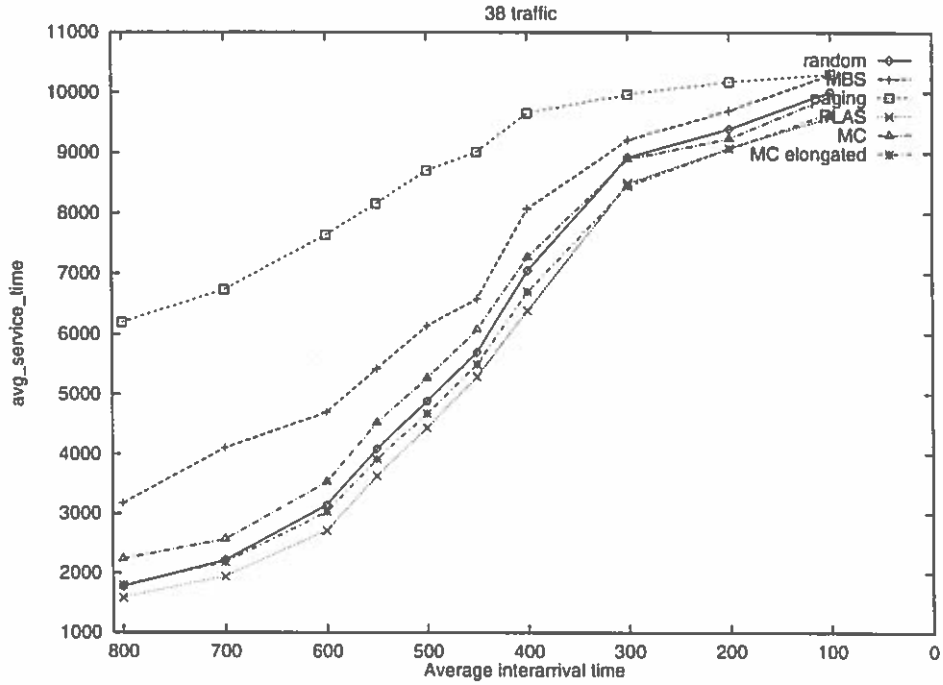


FIGURE 62. 80% I/O Traffic (Synthetic Workload)

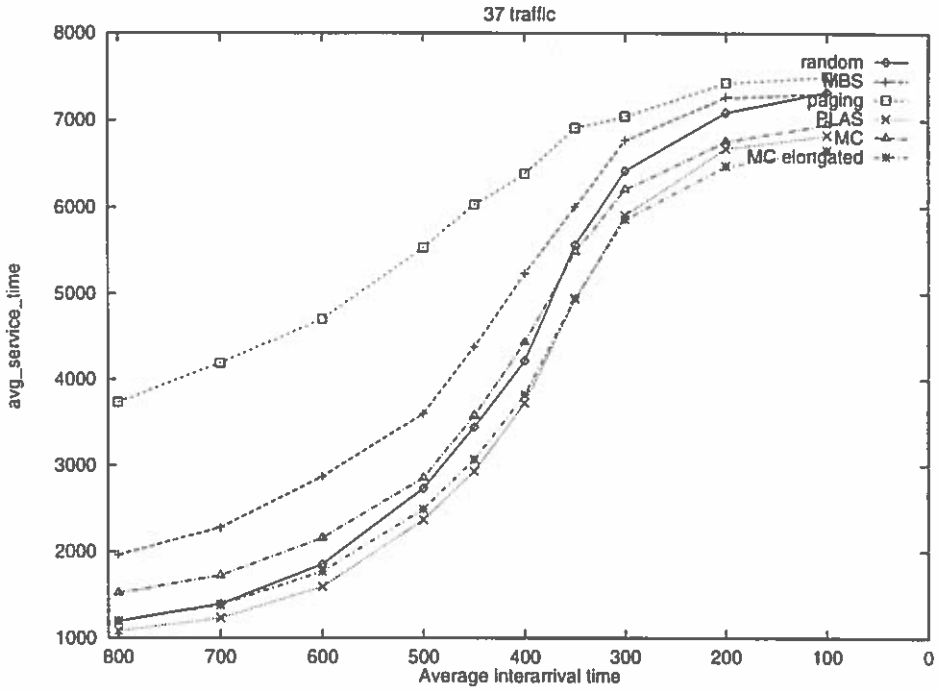


FIGURE 63. 60% I/O Traffic (Synthetic Workload)

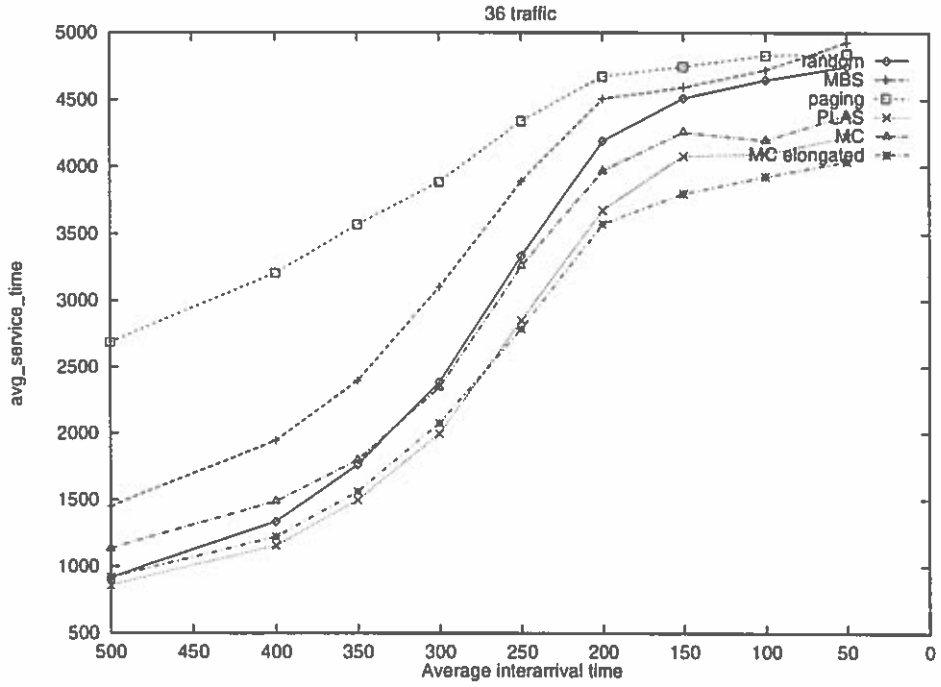


FIGURE 64. 40% I/O Traffic (Synthetic Workload)

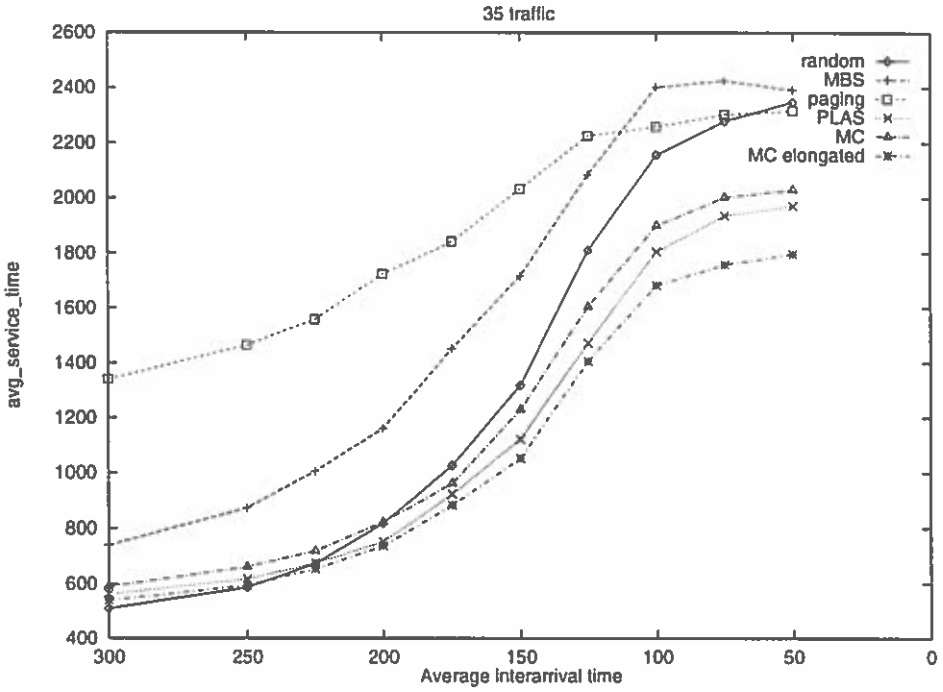


FIGURE 65. 20% I/O Traffic (Synthetic Workload)

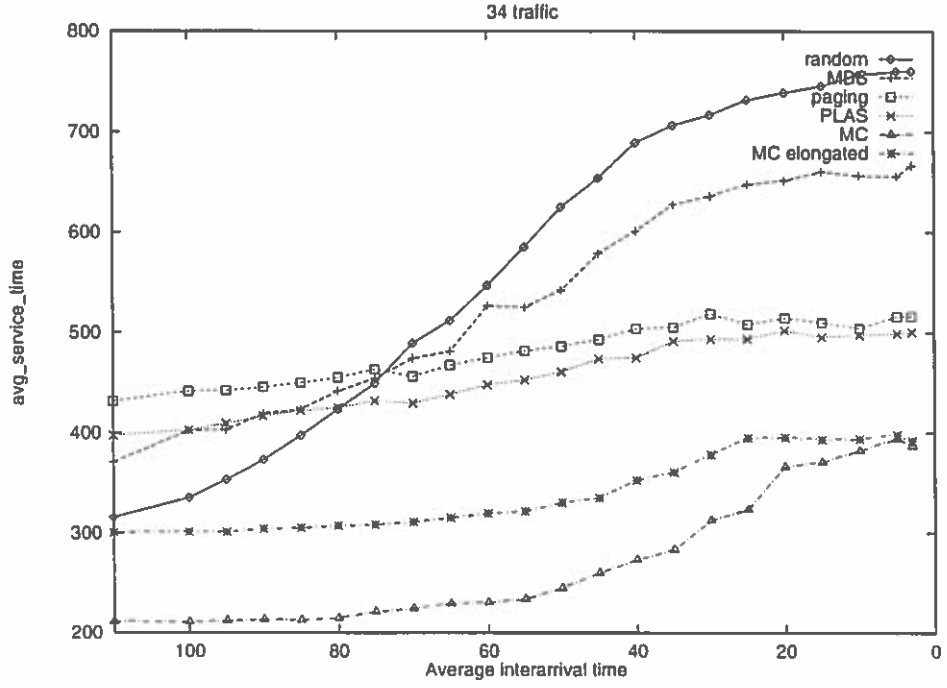


FIGURE 66. 0% I/O Traffic (Synthetic Workload)

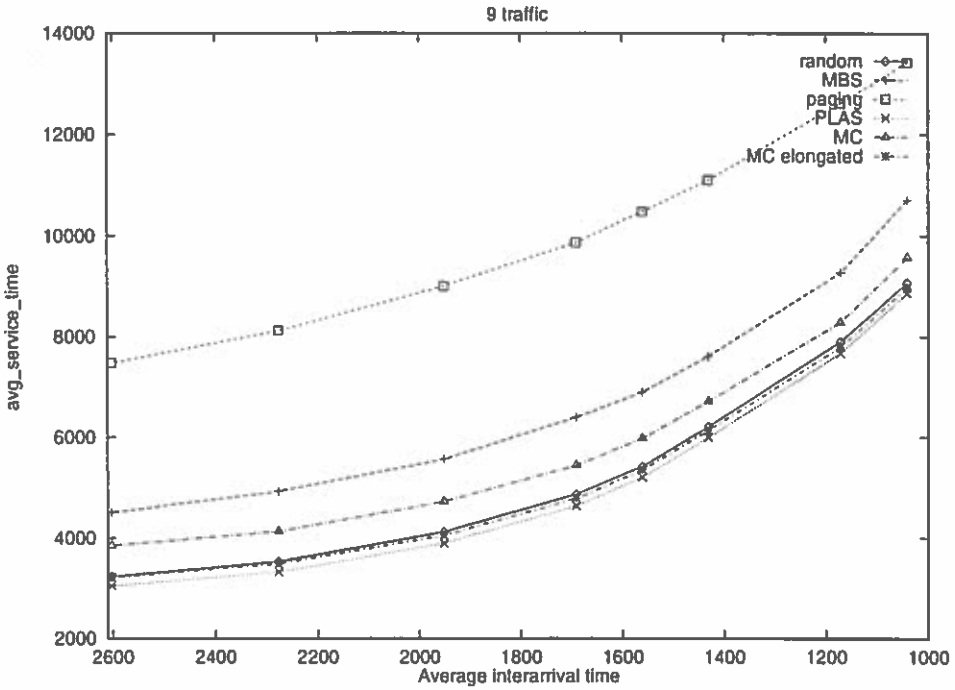


FIGURE 67. 100% I/O Traffic (SDSC Workload)

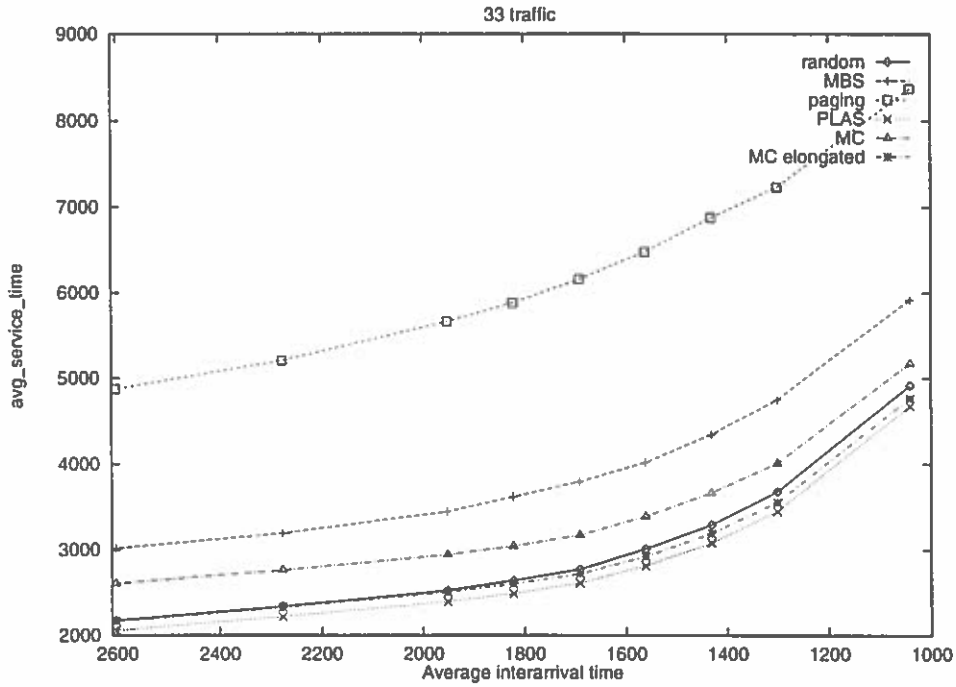


FIGURE 68. 75% I/O Traffic (SDSC Workload)

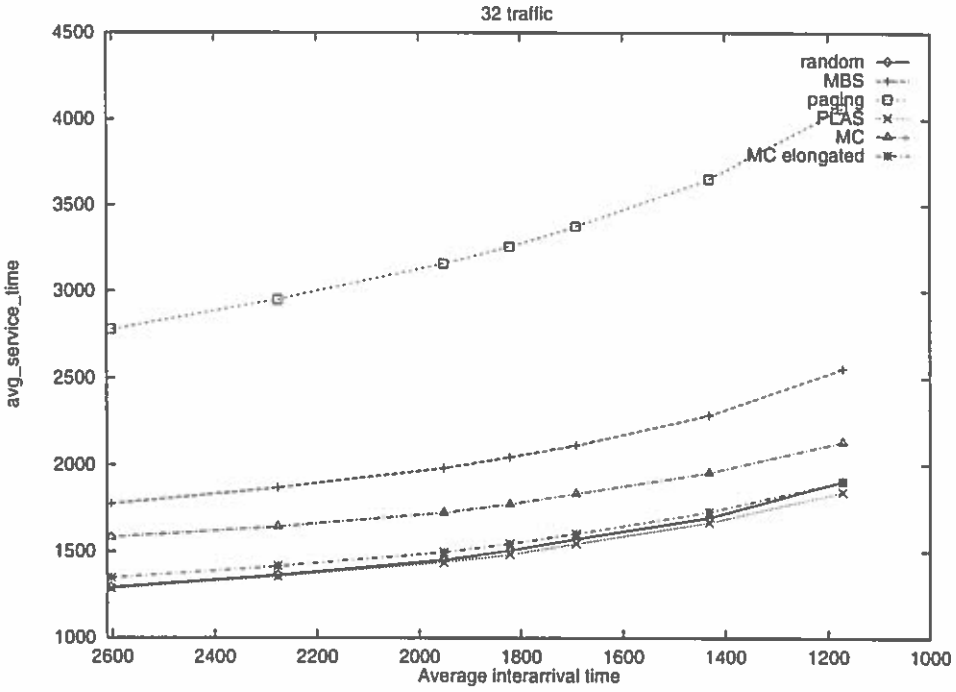


FIGURE 69. 50% I/O Traffic (SDSC Workload)

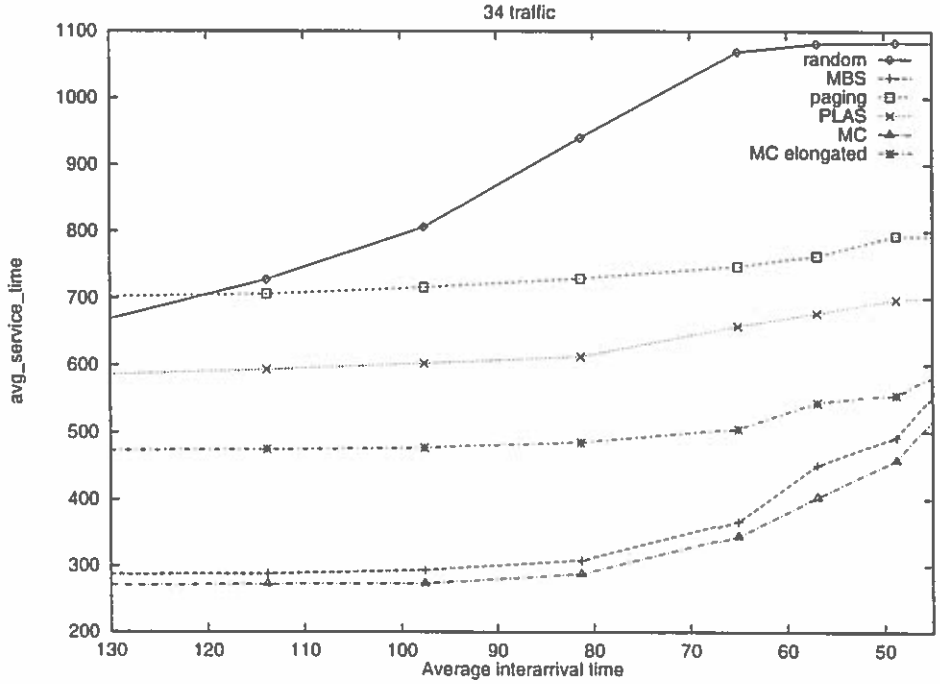


FIGURE 70. 0% I/O Traffic (SDSC Workload)

Table 9 summarizes average service time and ranking for the synthetic workload. Interarrival times are chosen such that system utilization is between 60% and 70%. While MC Elongated is not as good as PLAS for 100% I/O traffic or as good as MC at 0% I/O traffic, MC Elongated achieves the best average service time for 40% I/O traffic and 20% I/O traffic.

TABLE 9. Average Service Time (serv. t) and Ranking (r) for Different Ratios of I/O Traffic vs. Communication Traffic (Interarrival Times Are Chosen such that System Utilization Is Between 60% and 70%)

traffic ratio	100% I/O		80% I/O		60% I/O		40% I/O		20% I/O		0% I/O	
interarrival t.	500		400		350		250		125		20	
	serv. t	r	serv. t	r	serv. t	r	serv. t	r	serv. t	r	serv. t	r
MC Elongated	8529	3	6693.6	2	4948.7	2	2784.0	1	1405.0	1	395.3	2
PLAS	8140	1	6384.6	1	4929.5	1	2847.9	2	1470.9	2	501.9	3
MC	9076	4	7265.1	4	5481.7	3	3258.8	3	1603.4	3	366.0	1
Random	8488	2	7046.6	3	5563.9	4	3334.9	4	1810.0	4	738.9	6
MBS	9939	5	8072.3	5	6004.2	5	3890.3	5	2085.2	5	651.4	5
Paging	12096	6	9653.6	6	6911.8	6	4341.1	6	2225.7	6	514.5	4

The success of MC Elongated is due to its ability to achieve both compactness and balance. Figure 71 shows that the average balance factor of MC Elongated is much better (lower) than that of MC. Figure 72 shows that dispersal of MC Elongated is much better (lower) than that of PLAS. (While the service time graphs can be quite different for different traffic ratios, the dispersal and balance graphs look almost the same for all traffic ratios if interarrival times are chosen such that utilization levels are similar.)

General Observations

Sifting through hundreds of graphs, we made three important observations. First, varying system load has quite the opposite effect on balance and I/O inten-

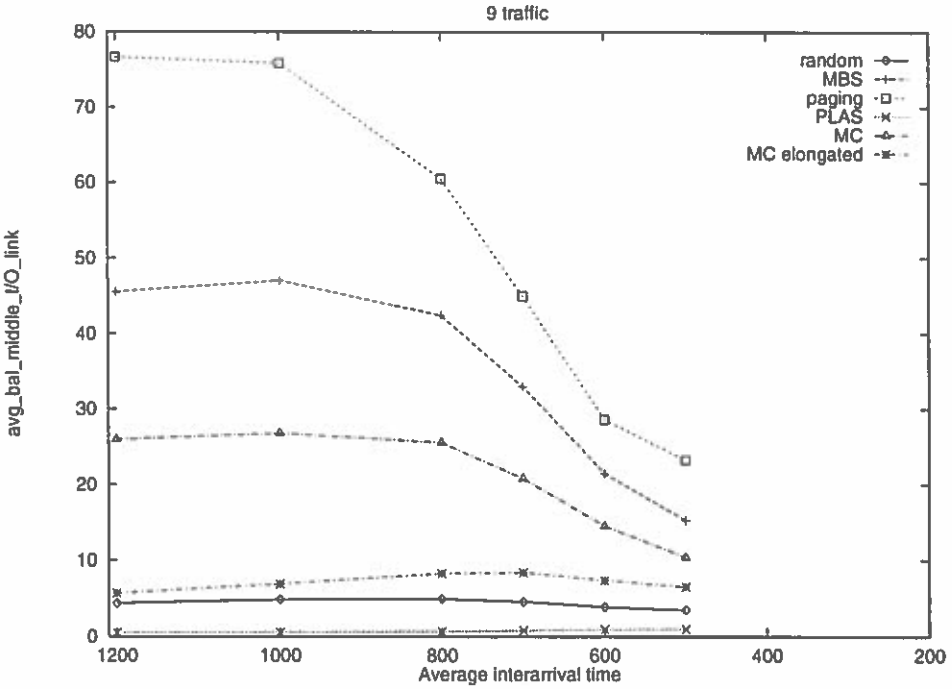


FIGURE 71. Average Balance Factor

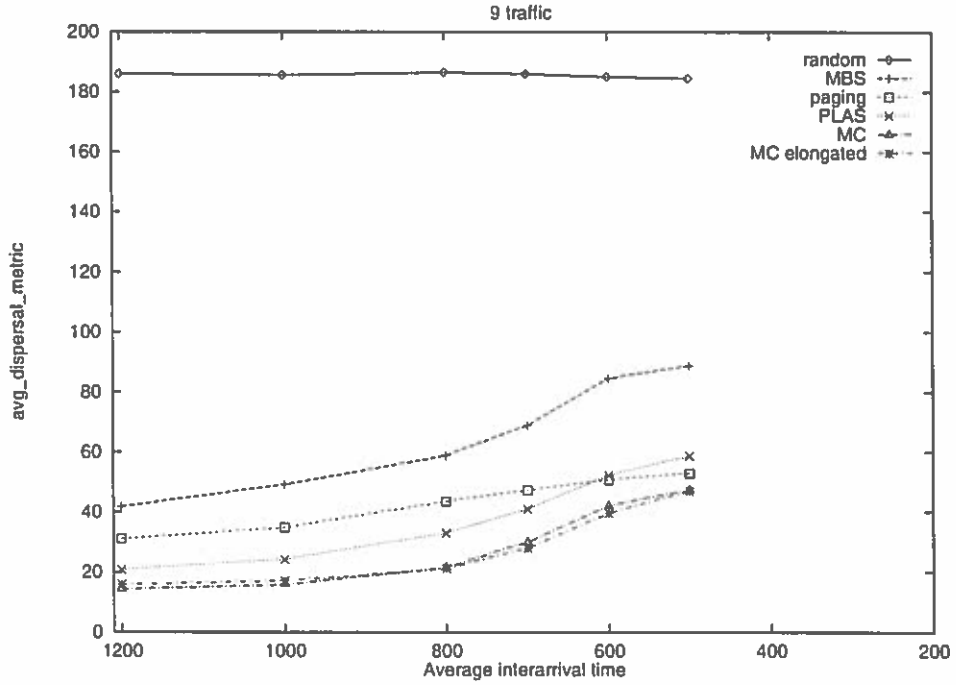


FIGURE 72. Average Dispersal Metric

sive workloads than it has on compactness and communication-intensive workloads. As the system load increases (interarrival time decreases, utilization increases), the average balance factor of Paging, MBS and MC decreases (see Figure 71). The fuller the system the more likely we get good balance for free. If the system is full, it is automatically balanced. This is also the reason why at high loads, relative improvements in service time decreases (see Figure 61). In other words: For I/O-intensive workloads, spatial layout (balance) is more important at lower loads.

However, as system load increases, dispersal increases (see Figure 72). It becomes harder to find compact allocations. Regarding service time (see Figure 66), it is interesting to note that dispersal gets more important as system load increases. Random and MBS have high dispersal. At low loads they get away with it since often there are no other jobs present to interfere with. However, as the load increases, their service time decreases much faster than that of the other strategies. In other words: for communication-intensive workloads, compactness is more important at higher loads.

Second, I/O traffic dominates over communication traffic with respect to network contention, service time and response time. Even at a traffic ratio of only 20% I/O traffic, the performance graphs (shapes, dependence on system load as described above) are more similar to those at a 100% I/O ratio than to those at a 0% I/O ratio.

Third, the increase in service time should not be underestimated. In all the strategies and experiments, we assumed unrestricted admission, i.e. a job is always allocated no matter how full the system or how good or bad the spatial layout determined by the allocation algorithm. This has the advantage of not wasting

compute power, by leaving compute nodes idle. However, as all the service time graphs show for all allocation strategies, the faster jobs arrive and the fuller we pack the system, the higher is the average service time of jobs due to communication interference and I/O hotspots. Besides applying a good allocation strategy that minimizes network contention, there probably is a point where it does not pay to pack yet another job into the system, because all the jobs get slowed down. It may be of advantage for all the jobs to restrict admission in order to avoid this increase in service time. This is an area of future work (see Chapter VIII).

CHAPTER VIII

CONCLUSIONS

As communication and I/O traffic increases on the interconnection network of high-performance parallel computers, *network contention* becomes a critical problem, drastically reducing effective data throughput. Minimizing network contention is crucial in order to achieve fast job response times and high system throughput.

While network contention can be affected by the resource management issue of processor allocation, previous processor allocation strategies have essentially ignored the I/O and communication demands of parallel applications and the resulting network contention. In this thesis, we have conducted a systematic analysis of the effects that the processor allocation strategy, through its decisions on the *spatial layout* of jobs, has on network contention; and we have designed and tested new processor allocation strategies that minimize network contention by being sensitive to communication and parallel I/O.

Our new parallel I/O- and communication-sensitive allocation strategies assign a set of compute nodes to each scheduled job such that (1) communication traffic of different jobs interfere as little as possible and (2) parallel I/O traffic is distributed as evenly as possible.

Summary

In Chapter IV, we focused on inter-job link contention due to communication between possibly dispersed compute nodes. We investigated the impact of spatial layout of jobs on inter-job link contention. We defined several *dispersal metrics* that measure the spatial layout of a given job's allocation in order to capture the degree of compactness of that allocation. Through dynamic simulation, we analyzed the relationship between spatial layout, network contention, and system throughput. Results motivated the need for *allocation compactness* in order to minimize communication-based network contention.

In Chapter V, we applied what we have learned to the design of *MC*, a new processor allocation strategy that is communication-sensitive. By allocating jobs as compactly as possible, MC was successful in minimizing inter-job link contention. Our performance evaluation shows improvements in average response time of up to a factor of 4.9 for communication-intensive jobs.

In Chapter VI, we focussed on traffic hotspots due to data transfer between compute nodes and I/O nodes. We analyzed the impact of spatial layout of jobs on traffic hotspots. Through graph theoretic analysis and dynamic simulation, we analyzed the relationship between spatial layout, traffic hotspots, and realizable I/O throughput. We proved theorems on the limitation of realizable I/O throughput due to network contention and on an upper bound on the number of I/O nodes that can be added to the system before the network saturates. Results motivated the need for careful spatial layout (*shape and location*) of compute nodes in order to minimize parallel I/O-based network contention.

In Chapter VII, we defined a new metric called balance factor and presented

PLAS, a new parallel I/O-sensitive allocation algorithm. PLAS alleviates traffic hotspots by concentrating on minimizing contention on the middle I/O link and by striving for balanced allocations. PLAS improved the response times of I/O-intensive jobs by up to a factor of 4.5. Finally, we designed *MC Elongated*, a new allocation algorithm that is both parallel I/O and communication-sensitive. MC Elongated strives for balance and allocation compactness at the same time. MC Elongated improved the performance under workloads that are both communication-intensive and parallel I/O-intensive by up to a factor of 2.9.

Contributions

We took processor allocation strategies one step beyond the existing state of the art. Non-contiguous allocation strategies had focussed on the *computation* demands of parallel applications and had solved the fragmentation problem. However, previous research had essentially ignored the *I/O* and *communication* demands of parallel applications and the resulting network contention.

The primary contributions of this work are the following. We summarize each contribution in greater detail below.

1. Analysis of the effects of spatial layout of jobs on network contention and implications on performance.

2. New concepts: compactness for communication, balance for I/O.

3. New metrics that are practical to use for performance evaluation and to guide allocation decisions.

4. New algorithms that are successful in minimizing network contention and

thus in improving performance.

First we give more details regarding communication-based network contention:

1. We gained a deeper understanding of inter-job link contention, continuing the work of Min and Mutka [40] and Enbody [9]. While they recognized that network contention is a dynamic phenomenon and thus very complex, we were the first to develop quantitative metrics to measure the degree of inter-job link contention.

2. Previous work has shown that (1) allocating jobs contiguously-only is an overly restrictive allocation constraint, wasting idle compute nodes, (2) allocating jobs non-contiguous-randomly fails to minimize network contention and (3) allocating jobs non-contiguously, but contiguous within blocks, is a hybrid strategy that performs better than any of the pure alternatives. Yet, none of these strategies adequately addressed the complex issue of network contention due communication.

   To address this shortcoming, we introduced the notion of compactness and the opposite notion, dispersal. Compactness and dispersal are needed to evaluate the extent to which non-contiguous allocations interfere with the communication of other jobs. Before our work, there was no precise notion of how to measure this.

3. We developed metrics to evaluate compactness. Our metrics – *nodes_affected*, *links_affected*, as well as distances and diameter – have high correlation with inter-job link contention. They are easily computed at allocation time and thus can be used within allocation algorithms; they are also easy and effi-

cient to compute at allocation time and thus can be used for performance evaluation.

4. We designed the MC strategy that strives for compactness. MC is very successful, achieving very compact allocations and thus reducing service time and improving average response time. We gained an understanding of the limitations of block-based or conservative approaches to non-contiguous allocation. They reap limited benefits because often the blocks do not get used or do not get re-combined. In the end the system ends up quite fragmented anyways. Paging based approaches are simpler and perform just as well. These are the foundation for MC, PLAS, and MC Elongated. As we saw they are also beneficial for I/O.

Next we give more details regarding I/O-based network contention:

1. We analyzed the limits of parallel I/O performance on mesh topologies with I/O nodes on one side. Network bandwidth per link is critically important. Even with an optimal spatial layout of jobs, the overall parallel I/O performance is limited to four times the network bandwidth per link under heavy (complete bipartite) I/O traffic.

2. We introduced the notion of balance with respect to a job's allocated compute nodes. This notion is needed to evaluate the extent to which spatial layout of jobs contributes to uneven distribution of I/O traffic.

3. We developed metrics to evaluate hotspots ($max\_contention$) and balance ($balance factor$).

4. We designed the PLAS strategy that strives for balance. We designed the MC Elongated strategy that strives for both balance and compactness.

The results of this research showed that spatial layout has a much greater effect on network contention than previously believed. We showed that communication-sensitive and I/O-sensitive processor allocation algorithms can be designed which minimize the degree of network contention, and thus significantly improve overall performance. The results of this work have the potential to influence policies and decisions made within all dimensions of a high-performance system: from application program design to architectural configuration decisions to operating system libraries and resource management policies. Our work makes a contribution towards efficient resource management of teraflops-scale computing systems.

Future Work

Two main areas for further research are scheduling and architecture. In the area of scheduling and allocation, it would be interesting to extend this research (1) to other network topologies, such as multi-dimensional meshes, tori and hypercubes; (2) to heterogeneous traffic where different jobs have different ratios of I/O traffic vs. communication traffic; and (3) to different workloads, e.g. workloads with a correlation of 0 between jobsize and number of I/O or communication messages sent.

One idea worth exploring is the design of heterogeneous allocation strategies that place I/O-intensive jobs differently than communication-intensive jobs. For example, it might be beneficial to allocate I/O-intensive jobs starting from the side where the I/O nodes are and using a strategy like PLAS, while allocating

communication-intensive jobs starting from the opposite side and using a strategy like MC.

Further attention needs to be given to *job scheduling*, the decision of which job to allocate next and whether to allocate it now. If admitting an additional job slows other jobs down (e.g. due to network saturation), limited admission may be beneficial. The admission decision could be based on how compact or balanced the proposed allocation would be. Alternatively, the total number of nodes allocated to say I/O-intensive jobs can be limited. It would be worthwhile to study strategies that try to achieve a good mix of currently allocated jobs with regards to their demands on the network. If admission is limited, it would be interesting to investigate "backfilling" otherwise idle compute nodes with jobs that do not do I/O, for example.

Finally, research problems arise in the area of architecture. It would be interesting to investigate the effect of other I/O node placement schemes such as those reported in [2, 34]. How should a parallel system be configured such that the network does not become the bottleneck?

This dissertation takes a concrete step at resolving several issues related to communication-based and I/O-based network contention and opens the door to a wealth of new challenging problems in the area of resource management for high-performance parallel systems.

BIBLIOGRAPHY

[1] A. Al-Dhelaan and B. Bose. A new strategy for processor allocation in an nCUBE multiprocessor. In *Proceedings of the International Phoenix Conference on Computers and Communication*, pages 114–118, March 1989.

[2] M. Bae and B. Bose. Resource placement in torus-based networks. *IEEE Transactions on Computers*, 46(10):1083–1092, October 1997.

[3] Rajive Bagrodia, Andrew Chien, Yarson Hsu, and Daniel Reed. Input/output: Instrumentation, characterization, modeling and management policy. Technical Report CCSF-41, Scalable I/O Initiative, Caltech Concurrent Supercomputing Facilities, Caltech, 1994.

[4] David H. Bailey, Eric Barszcz, Leonardo Dagum, and Horst D. Simon. NAS parallel benchmark results. *IEEE Parallel and Distributed Technology*, 1(1):43–51, February 1993.

[5] Sandra Johnson Baylor, Caroline Benveniste, and Yarsun Hsu. Performance evaluation of a massively parallel I/O subsystem. In Ravi Jain, John Werth, and James C. Browne, editors, *Input/Output in Parallel and Distributed Computer Systems*, volume 362 of *The Kluwer International Series in Engineering and Computer Science*, chapter 13, pages 293–311. Kluwer Academic Publishers, 1996.

[6] S. H. Bokhari and D. M. Nicol. Balancing contention and synchronization on the Intel Paragon. Technical Report 96-54, ICASE, 1996.

[7] B. Bose, B. Broeg, Y. Kwon, and Y. Ashir. Lee distance and topological properties of k-ary n-cubes. Technical Report 93-60-11, Oregon State University, 1993.

[8] M. Chen and K. G. Shin. Processor allocation in an nCUBE multiprocessor using Gray codes. *IEEE Transactions on Computers*, C-36(12):1396–1407, December 1987.

[9] Suresh Chittor and Richard Enbody. Performance evaluation of mesh-connected wormhole-routed networks for interprocessor communication in multicomputers. In *Proceedings of Supercomputing '90*, pages 647–656, 1990.

[10] Y. Cho, M. Winslett, M. Subramaniam, Y. Chen, S. Kuo, and K. E. Seamons. Exploiting local data in parallel array I/O on a practical network of workstations. In *Proceedings of the 5th Workshop on I/O in Parallel and Distributed Systems, SC '97*, pages 1–13, 1997.

[11] P. Chuang and N. Tzeng. An efficient submesh allocation strategy for mesh computer systems. In *Proceedings of the International Conference on Distributed Computer Systems*, pages 256–263, 1991.

[12] R. Covington, S. Dwarkadas, J. Jump, J. Sinclair, and S. Madala. The efficient simulation of parallel computer system. *International Journal in Computer Simulations*, 1:31–58, 1991.

[13] Phyllis E. Crandall, Ruth A. Aydt, Andrew A. Chien, and Daniel A. Reed. Input/output characteristics of scalable parallel applications. In *Proceedings of Supercomputing '95*, San Diego, CA, December 1995. IEEE Computer Society Press.

[14] J. Ding and L. N. Bhuyan. An adaptive submesh allocation strategy for two-dimensional mesh-connected systems. In *Proceedings of International Conference on Parallel Processing*, pages 193–200, 1993.

[15] Andreas Eberhart. The WARP simulator for wormhole-routed 2D mesh networks. Technical report, Portland State University, 1994.

[16] Andreas Eberhart and Jingke Li. Contention-free communication scheduling on 2D meshes. In *Proceedings International Conference on Parallel Processing*, 1996.

[17] B. Everitt. *Cluster analysis.* Halsted Press, 1993.

[18] D. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In *Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '98*, 1998.

[19] D. G. Feitelson. A survey of scheduling in multiprogrammed parallel systems. Technical Report RC 19790 (87657), IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, October 1994.

[20] Dror G. Feitelson, Peter F. Corbett, Sandra Johnson Baylor, and Yarson Hsu. Parallel I/O subsystems in massively parallel supercomputers. *IEEE Parallel and Distributed Technology*, 3(3):33–47, Fall 1995.

[21] M. L. Fulgham and L. Snyder. Performance of Chaos and oblivious routers under non-uniform traffic. Technical report, University of Washington, 1993.

[22] Sharad Garg. Parallel I/O architecture of the first ASCI TFLOPS machine. In *Proceedings of Intel Supercomputer Users Group*, 1997.

[23] Sharad Garg. I/O measurements on Intel TFLOPS with 18 I/O nodes and 400 compute nodes. Personal communication, 1998.

[24] Sharad Garg, Robert Godley, Richard Griffiths, Andrew Pfiffer, Terry Prickett, David Robboy, Stan Smith, T. Mack Stallcup, and Stephen Zeisset. Achieving large scale parallelism through operating system resource management on the Intel TFLOPS supercomputer. *Intel Technology Journal*, 1st quarter 1998.

[25] H.-U. Heiss. Processor management in two-dimensional grid-architectures. Technical Report 20/92, Universität Karlsruhe, 1992.

[26] R. Jain, K. Somalwar, J. Werth, and J. C. Browne. Heuristics for scheduling I/O operations. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):310–320, March 1997.

[27] C. Koelbel, D. Loveman, R. Schreiber, G. Steele, and M. Zosel. *High Performance Fortran Handbook.* MIT Press, 1994.

[28] David Kotz. Disk-directed I/O for MIMD multiprocessors. In *Proceedings of the 1994 Symposium on Operating Systems Design and Implementation*, pages 61–74. USENIX Association, November 1994. Updated as Dartmouth TR PCS-TR94-226 on November 8, 1994.

[29] David Kotz and Nils Nieuwejaar. Dynamic file-access characteristics of a production parallel scientific workload. In *Proceedings of Supercomputing '94*, pages 640–649, Washington, DC, November 1994. IEEE Computer Society Press.

[30] P. Krueger, T. Lai, and V. A. Dixit-Radiya. Job scheduling is more important than processor allocation for hypercube computers. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):488–497, May 1994.

[31] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing.* Benjamin Cummings Publishing Company, Inc., 1994.

[32] K. Li and K.-H. Cheng. A two-dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system. *Journal of Parallel and Distributed Computing*, 12:79–83, 1991.

[33] W. Liu, V. Lo, K. Windisch, and B. Nitzberg. Non-contiguous processor allocation algorithms for distributed memory multicomputers. In *Proceedings of Supercomputing '94*, pages 227–236, 1994. Best student paper award, also in IEEE Transactions on Parallel and Distributed Systems, July 1997.

[34] Marilynn Livingston and Quentin Stout. Distributing resources in hypercube computers. In *Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications*, 1988.

[35] V. M. Lo, K. Windisch, W. Liu, and B. Nitzberg. Non-contiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 8(7):712–726, July 1997.

[36] Virginia Lo, Jens Mache, and Kurt Windisch. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In *Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '98*, 1998.

[37] Jens Mache and Virginia Lo. The effects of dispersal on message-passing contention in processor allocation strategies. In *Proceedings of the 3rd Joint Conference on Information Sciences, Volume 3, Sessions on Parallel and Distributed Processing*, 1997.

[38] Jens Mache, Virginia Lo, and Kurt Windisch. Minimizing message-passing contention in fragmentation-free processor allocation. In *Proceedings of the 10th International Conference on Parallel and Distributed Computing Systems*, 1997.

[39] Ethan L. Miller and Randy H. Katz. Input/output behavior of supercomputer applications. In *Proceedings of Supercomputing '91*, pages 567–576, Albuquerque, NM, November 1991. IEEE Computer Society Press.

[40] D. Min and M. W. Mutka. A multipath contention model for analyzing job interaction in 2-D mesh multicomputers. In *Proceedings of the 8th International Parallel Processing Symposium*, pages 744–751, April 1994.

[41] Bernd Mohr. Processor allocation on Cray T3E. Personal communication, 1998.

[42] S. Q. Moore and L. M. Ni. The effects of network contention on processor allocation strategies. Technical Report MSU-CPS-ACS-106, Michigan State University, 1995.

[43] Csaba A. Moritz and Matthew I. Frank. LoGPC: Modeling network contention in message-passing programs. In *Proceedings of Sigmetrics*, 1998.

[44] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Transactions on Computers*, pages 62–76, February 1993.

[45] Bill Nitzberg. Processor allocation on Intel Paragon. Personal communication, 1998.

[46] William J. Nitzberg. *Collective Parallel I/O.* PhD thesis, Department of Computer and Information Science, University of Oregon, December 1995.

[47] Barbara K. Pasquale and George C. Polyzos. Dynamic I/O characterization of I/O intensive scientific applications. In *Proceedings of Supercomputing '94*, pages 660–669, Washington, DC, November 1994. IEEE Computer Society Press.

[48] James C. T. Pool and Paul Messina. Scalable I/O initiative. http://www.cacr.caltech.edu/SIO/.

[49] Apratim Purakayastha, Carla Schlatter Ellis, David Kotz, Nils Nieuwejaar, and Michael Best. Characterizing parallel file-access patterns on a large-scale multiprocessor. In *Proceedings of the Ninth International Parallel Processing Symposium*, pages 165–172. IEEE Computer Society Press, April 1995.

[50] M. Rosenkrantz, D. Schneider, R. Leibensperger, M. Shore, and J. Zollweg. Requirements of the cornell theory center for resource management and process scheduling. In *Proceedings of the 1st Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '95*, 1995.

[51] J. Subhlok, T. Gross, and T. Suzuoka. Impacts of job mix on optimizations for space sharing schedulers. In *Proceedings of Supercomputing '96*, 1996.

[52] K. Suzaki, H. Tanuma, S. Hirano, Y. Ichisugi, C. Connelly, and M. Tukamoto. Multi-tasking method on parallel computers which combines a contiguous and a non-contiguous processor partitioning algorithm. In *Lecture Notes in Computer Science 1184*, pages 641–650, 1996.

[53] M. Wan, R. Moore, G. Kremenek, and K. Steube. A batch scheduler for the Intel Paragon MPP system with a non-contiguous node allocation algorithm. In *Proceedings of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '96*, 1996.

[54] Mu-Cheng Wang, Howard Jay Siegel, Mark A. Nichols, and Seth Abraham. Using a multipath network for reducing the effects of hot spots. *IEEE Transactions on Parallel and Distributed Systems*, 6(3):252–268, March 1995.

[55] K. Windisch, V. Lo, and B. Bose. Contiguous and non-contiguous processor allocation algorithms for k-ary n-cubes. In *Proceedings of the International Conference on Parallel Processing*, 1995.

[56] K. Windisch, V. Lo, D. Feitelson, B. Nitzberg, and R. Moore. A comparison of workload traces from two production parallel machines. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation*, 1996.

[57] K. Windisch, J. V. Miller, and V. Lo. ProcSimity: an experimental tool for processor allocation and scheduling in highly parallel systems. In *Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computation*, pages 414–421, 1995.

[58] Y. Zhu. Efficient processor allocation strategies for mesh-connected parallel computers. *Journal of Parallel and Distributed Computing*, 16:328–337, 1992.