Performance Analysis of an Adaptive Symmetric Broadcast Load Balancing Algorithm on the Hypercube

Sajal K. Das and Daniel J. Harvey
Department of Computer Sciences
University of North Texas
P.O. Box 13886
Denton, TX 76203-6886
E-mail:{das,harvey}@cs.unt.edu

Abstract

As distributed computing networks become more and more popular, it is important to insure that the amount of processing performed by the various processors (or nodes) is as balanced as possible. More specifically, it is desirable to prevent, if possible, the condition where one node is overloaded with a backlog of jobs to be processed while another processor is lightly loaded or idle.

Among numerous algorithms developed to address the load balancing problem, a unique approach utilizing what is known as a symmetric broadcast network was originally proposed by Das et al. [1, 2, 3]. By defining robust communication patterns between the nodes of a network in a topology-independent manner, very promising and original balancing algorithms have been derived.

This paper proposes a refined algorithm that utilizes the symmetric broadcast approach adapted for use on a hypercube multicomputer. By extensive simulation studies using a Poisson distribution of job generation (i.e. workload) we compare the proposed algorithm to a variety of other existing load balancing algorithms.

The empirical results of our experiments show that the symmetric broadcast algorithm is superior in being able to balance system load and to minimize processor idle time.

Key words: network, topology, hypercube, system load, job migration, spanning binomial tree, homogeneous network, Poisson distribution.

1 Introduction

In order to maximize the usefulness of a multicomputer system, it is essential to even the load between the processors of the network. in [13], it was determined that without load balancing, even when a multicomputer network is relatively busy, it is likely that some processors are heavily loaded while others are lightly loaded or idle.

The load balancing problem is closely related to scheduling and resource allocation. [14] presents many papers that describe this relationship. For example, load balancing, scheduling, and resource allocation can be static [15, 16] or dynamic. Static allocation relates to decisions made at compile

time. If the static approach is to be used, compile time programming tools are necessary to adequately estimate the required resources [17].

Dynamic algorithms [19, 20], on the other hand, allocate/reallocate resources at run time in accordance with system parameters that are maintained. For example, These parameters determine when jobs can be migrated and account for the overhead involved in such a transfer [21]. Determining which parameters are to be maintained and how they are to be broadcast are important design considerations. Many of these issues are resolved based upon the objectives of the distributed scheduling policies [18, 22].

In the literature, (i.e. [4]), load balancing algorithms are classified according to their operational characteristics, the Symmetric Broadcast Algorithm that we propose can be classified as follows:

Adaptive The algorithm adapts its performance based on the total load level (the total number of jobs queued for processing in the system).

Symmetrically Initiated Both senders and receivers can initiate load balancing activities.

Stable The algorithm under certain situations does not burden the network with excessive balancing traffic. Under light system loads, such traffic can occur when an unstable algorithm at a particular node repeatedly looks to other nodes for jobs to process. Similarly, excessive traffic could also occur in the thrashing of jobs from process to process when the system load is heavy.

Effective System performance does not degrade when the algorithm is operating.

This paper considers general purpose distributed-memory parallel computers that are connected by a point to point network topology. The nodes of the network communicate using message passing. Responsibility for load balancing activities are decentralized, or spread throughout the nodes of the network. The load at a particular processor is determined by the length of its local job queue. For simplicity, we assume that the network is homogeneous and any job can be processed by any node. However jobs cannot be rerouted once execution begins.

The symmetric broadcast approach to load balancing, proposed by Das et al. [1, 2, 3] is topology independent. We modify the approach to adapt the concept for use on hypercubes. Analogous adaptations can easily be devised for use on other network topologies.

The proposed algorithm is analyzed, using simulation, via a parallel virtual machine that has been constructed using the PVM package. Popular load balancing algorithms that are compared include the Random [5], Gradient [6, 7], Sender Initiated [8], Receiver Initiated [8], and Adaptive Contracting methods [8]. The algorithm is also compared to a network model using no balancing at all.

The empirical results gathered by our analysis show that load balancing achieved using the symmetric broadcast network is superior to that of the other algorithms that are analyzed. Further adaptations of this algorithm to other network topologies should provide a general solution to a wide variety of applications.

This paper is organized as follows. Section 2 discusses alternate approaches to measuring load balancing algorithm performance and introduces the various algorithms that are compared. Section 3 describes the refinements made to adapt the symmetric broadcast network to the hypercube and provides an example. Section 4 provides a detailed description of the implementation. Section 5

describes the performance analysis and discusses the results. Section 6 concludes the paper with possible direction for future research.

2 Preliminaries

2.1 Related Work

Among various approaches that have been used in the literature for comparing load balancing algorithms, three categories of analysis predominate. These are: (i) mathematical modeling, (ii) solving well known problems in a parallel environment, (iii) simulation. For example, in [9], the probability of load balancing success is computed analytically. In [10], several load balancing methods are compared by implementing Fibonacci number generation, the N-Queens problem, and the fifteen puzzle on a network. Many papers have been written that employ the simulation approach to analysis (i.e. [11]).

In this paper we choose the simulation approach, using synthetical loads. Actual load conditions in a network defy accurate mathematical modeling. Solving particular problems tend to generate loads that are not typical of actual loads.

Many load balancing algorithms that are compared are very susceptible to the choice of system thresholds. In [12], a study was conducted relating to proper threshold selection. This information was helpful for optimizing the Symmetric Broadcast algorithm that will be described. The existing algorithms that we compare in this paper are briefly described below:

Random [5]

Jobs are randomly distributed between nodes.

As jobs are generated, if the system load of a given node is above a designated threshold, the jobs are randomly distributed between the originating and neighbor nodes. Once a job originating at one node has been received at another node, it is processed. Therefore, job migration is not allowed. Single distribution messages can contain multiple jobs when more than one job is to be sent from one node to another.

Gradient [6, 7]

Jobs proceed from overloaded to lightly loaded nodes. This is accomplished by a system wide gradient that is maintained. The gradient is constructed as follows:

Each node maintains a load status flag. The value stored in this flag determines whether the node is overloaded, lightly loaded, or moderately loaded. The setting of this value depends on system thresholds.

An array (PRESSURE) is also maintained at each node. This array has one entry that corresponds to each of its neighbors. Each of these entries contain the pressure (minimum number of communication "jumps" to the nearest lightly loaded node) if a job is to be routed to the neighbor that corresponds to this entry.

The pressure of any node is zero, if that node is lightly loaded. Otherwise, a node's pressure

is calculated in accordance with the following formula:

$$min(PRESSURE[n] \text{ for each neighbor node, } n) + 1$$

Whenever the pressure of a node changes, that pressure is broadcast to all of its neighbors. Note that because of network dynamics, the pressure value is only an approximation to the true system value.

Under the gradient algorithm, a job can migrate many times before it is finally processed.

Receiver Initiated [5, 10]

Load balancing is triggered by a lightly loaded node. If a given node has a load value below the system threshold value, it broadcasts a job request message to its neighbors. The node's job queue length is "piggy backed" to the request message.

Upon receipt of this message, each neighbor node compares its job queue length to that of the requesting node. If the local queue size is greater, the neighbor node replies with a single job.

To prevent instability in light system load conditions, a time-out of one second is introduced to wait for job replies. More specifically, the node will wait one second before initiating another request for jobs.

It is possible for a job to be migrated multiple times using this algorithm before being processed.

Sender Initiated [5, 10]

Load balancing is initiated when nodes become overloaded. To prevent instability under heavy system loads, each node exchanges load information with its neighbors. Load values are exchanged when a local job queue goes from 2^{x+1} to 2^x in length (or visa versa). In this way, exchange of load information occurs less and less often as the system load increases.

When jobs are generated, they are distributed to lightly loaded neighbors. Once a job is received from a neighbor node, it is processed. Multiple job migrations are not allowed.

Adaptive Contracting [5, 10]

When jobs are generated, the originating node, in parallel, distributes bids to its neighbor nodes. The neighbor nodes respond to this bid with a message containing the number of jobs that it has in its local queue.

The originating node will then distribute jobs to those neighbors that have system loads smaller than an amount determined by a system threshold value. The number of jobs distributed are computed to divide the jobs equally between the originating node and its lightly loaded neighbors.

2.2 Definitions

The symmetric broadcast network(SBN) approach to load balancing is described in detail in [1, 2, 3]. Figure 1 provides an example of two communication paths in such a network. For completeness, the definition that follows describes how to construct an SBN.

Figure 1: Examples of Symmetric Broadcast Networks on a network of 8 nodes

An SBN is a multistage interconnection network for communication among processors. Each stage of an SBN of dimension d has $P = 2^d$ processors. The SBN is constructed recursively as follows:

1. A single node forms the basis for a network of dimension zero.
2. An SBN of dimension d is constructed from a pair of SBN networks of dimension $d-1$ by adding one additional communication stage and additional required inter computer connections. Specifically: (i) Node i, in Stage 0, is connected to node $j = (i + P/2) \bmod P$ of Stage 1 where $0 \leq i \leq P-1$. (ii) Node j in Stage 1 is connected to the node that was the Stage 0 successor of node i in the SBN of dimension $d-1$.

The idea behind the symmetric broadcast approach is to define unique communication patterns between all of the nodes in the multicomputer network in such a way that:

- In a network of P processors, starting with any node, n, there are $\log P$ stages of communication.
- Each node of the network appears once in the $\log P$ stages of communication.
- When a particular node initiates load balancing activity, it communicates with the neighbors that are defined by stage 1 of the communication path.
- The successors and predecessors to a given node of the network are uniquely defined by specifying the node that originated the communication and indicating which stage of communication is being processed.

As an example, consider a network of eight processors. Shown in Figure 1 are two communication paths. A path in Fig1.(b) can be obtained from a path in Fig1.(a) by applying an exclusive or operation to each node Fig1.(a) with 5 (the originating node). Similarly, a unique symmetric communication path can be constructed starting with any node of the network. Note that the network of eight nodes shown in Figure 1 has $3 = \log_2 8$ stages of communication. In this network, a node in the first stage always contains one successor node; while nodes in the subsequent stages contain a pair of successors.

Figure 1 is but one example of a symmetric broadcast network. Many other patterns can be devised depending upon the network topology. As we will see, the symmetric broadcast network for a hypercube utilizes a modified binomial spanning tree.

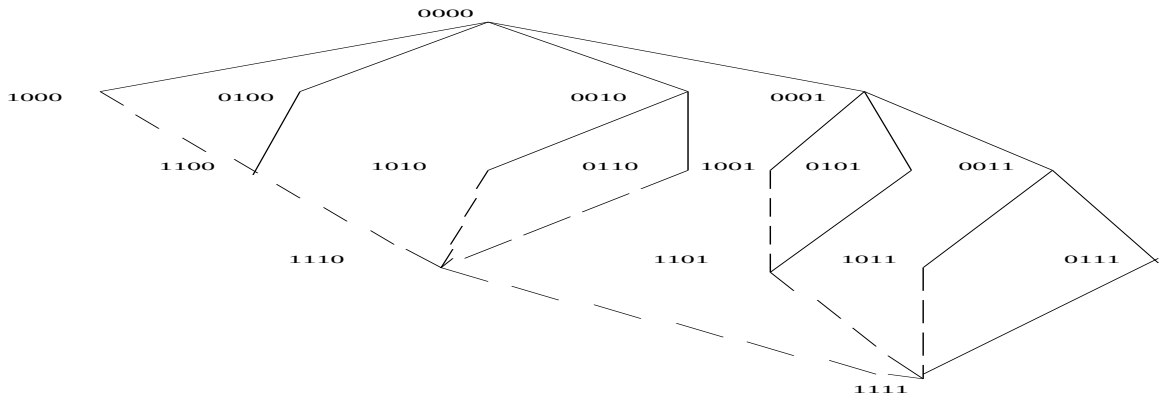


Figure 2: Binomial Spanning Tree as a Hypercube Symmetric Broadcast Network

3 Symmetric Broadcast Network

3.1 Adaptations for Hypercubes

In this paper we adapt the SBN just defined for use on the hypercube topology. This refinement is needed so that messages can be efficiently sent from one stage to the next without requiring extra communication "jumps".

The SBN that is used on a hypercube configuration utilizes a spanning binomial tree. An example of such a tree is shown in Figure 2 for a hypercube of 16 processors. Note that the solid lines represent the nodes of the spanning binomial tree. The dotted line connections are important for implementing the load balancing algorithm that will be described. Note that the characteristics presented in section 2 for an SBN are preserved. In addition, the modified binomial spanning tree has the following important characteristics:

- The modified binomial spanning tree insures that all successor and predecessor nodes at any communication stage are adjacent nodes in the hypercube.
- For every originating node, there is one single final node.
- Some nodes have multiple predecessors. If all of the nodes are numbered using a binary bit pattern of d bits, the number of predecessors can be determined by counting the number of leftmost 1 bits of a given node's binary pattern. Nodes with a 0 in the most significant bit have single predecessor. There are 2^{d-2} nodes that have more than one predecessor.

3.2 Algorithm Description

We will now describe the load balancing algorithm that makes use of the symmetric broadcast network just described.

There are two categories of load balancing messages processed by the network. The first category is the balancing request message. This message is sent through the network to indicate that the system load is out of balance. As this message is routed from stage to stage, the cumulative total of queued network jobs is computed. The balancing message is not routed from one node to its successors until all of the balancing messages have been gathered from its predecessors.

When the final single node at the last stage of communication gathers all of the balancing messages from its predecessors, the total system load is known with a high degree of accuracy. This information is then broadcast via distribution messages back through the symmetric network.

It takes an average of $O(\log P)$ time to process a load balancing request if we assume that communication from one node to its neighbors is completed in $O(1)$ time. However, it is possible that multiple balancing requests can be processed simultaneously. Therefore a worst case time of $O(\log^2 P)$ could be required. To reduce message traffic, a node will not initiate additional balancing requests until all previous balancing requests that have passed through this node have been completely processed.

The second message category of load balancing messages are job distribution messages. This message category is used for two purposes.

Firstly, job distribution messages are used to route the current total load level throughout the system. This is the final step of processing a load balancing request. Each node, upon receipt of such a message, updates its estimate of the average number of jobs queued per node in the network. This estimate is called the *system load level*. Various threshold values (described below) are then updated as well. The routing of job distribution messages is routed through the network using the symmetric broadcast approach, in the same manner as described for the load balancing messages.

Secondly, job distribution messages are used to pass excess jobs from node to node. This action can occur whenever a node has more jobs than its maximum threshold value. It can also be a response to a predecessor's need for jobs. This need is embedded in load balancing requests and in distributions responding to these requests. Distribution messages of this type do not have to be gathered and routed as previously described.

3.3 An Example

Shown in Figure 3 are the symmetric broadcast communication paths originating from node 3 and connecting to the other nodes of the network. Table 1 indicates, for each node, the system load level, the number of locally queued jobs, the minimum threshold, and the maximum threshold.

Note that node 3 has less jobs queued locally than its minimum threshold value. Therefore, a load balancing request is necessary. The request is sent to the successor nodes 2, 1, and 7. Node 3 includes its local queue size in the request message sent to its initial successor (node 2). Also included in all of the request messages is the need of 1 job from each neighbor. This need is computed by using the formula:

$$\text{need} = (\text{System load} - \text{Queue size})/(\text{number of neighbors})$$

Node 7 can immediately respond to this need because it has more jobs queued than indicated by its system load level. The results from the above action is shown in Table 2.

The balance request, received by nodes 7, 1, and 2 are forwarded to the successor nodes at the next level. Node 2 adds the size of jobs queued for processing at node 3 to the size of its own local queue and routes this total to node 0. Node 0 responds to the need of node 2 for jobs by distributing one job to node 2. Table 3 below shows the updated system parameters after completing this step.

Node 5 gathers the load balance requests from both of its predecessor nodes, 1 and 7, before for-

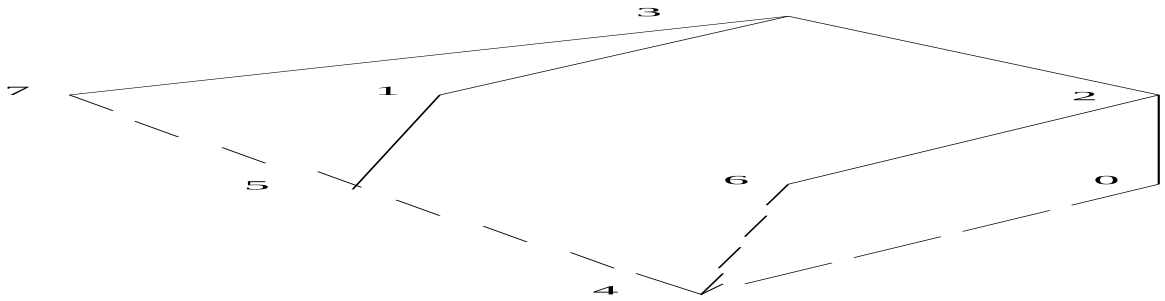


Figure 3: Binomial Spanning Tree as a Hypercube Symmetric Broadcast Network

Table 1: Initial Load in an 8 Node Symmetric Network

Nodes	0	1	2	3	4	5	6	7
Load	6	6	6	6	12	12	12	12
Queue	7	3	2	1	7	5	10	14
Min	2	2	2	2	2	2	2	2
Max	10	10	10	10	10	10	10	10

Table 2: Load after Balance Request Received by Nodes 7, 1, and 2

Nodes	0	1	2	3	4	5	6	7
Load	6	6	6	6	12	12	12	12
Queue	7	3	2	2	7	5	10	13
Min	2	2	2	2	2	2	2	2
Max	10	10	10	10	10	10	10	10

Table 3: Load after Balance Request Received by Nodes 5, 6, and 0

Nodes	0	1	2	3	4	5	6	7
Load	6	6	6	6	12	12	12	12
Queue	6	3	3	2	7	5	10	13
Min	2	2	2	2	2	2	2	2
Max	10	10	10	10	10	10	10	10

Table 4: Final Load after Balancing Algorithm has Completed

Nodes	0	1	2	3	4	5	6	7
Load	6	6	6	6	6	6	6	6
Queue	6	6	6	7	6	6	6	6
Min	2	2	2	2	2	2	2	2
Max	10	10	10	10	10	10	10	10

warding the balance request to node 7. The total number of jobs that are queued for processing continue to accumulate.

When node 4 receives the load balance request from all of its predecessors, (nodes 5, 6, and 0), it can compute the updated system load(S). Using this value and the network diameter(D), the minimum threshold(m) and maximum threshold(M) values are computed as well. These computations are shown in the equations below:

S =Jobs Queued/Total Processors $= 6$

$M = S + 2^{S/D} = 6 + 2^{6/3} = 10$

$m = \min(S/2, 2) = 2$

The distribution messages are returned through the network so that all of the nodes can update their system threshold values. Excess jobs are appropriately distributed as well.

The distributions to arrive at the final job distribution are: one job from node 4 to node 0, one job from node 0 to node 2, four jobs from from node 6 to node 2, one job from node 7 to node 5, two jobs from node 2 to node 3, six jobs from node 7 to node 3, and three jobs from node 3 to node 1. Table 4 shows the result of processing this load balancing request. As can be seen, the network load is very well balanced.

3.4 Thresholds

Three thresholds are defined to control the load balancing operation. More specifically, a *minimum threshold*, *maximum threshold*, and *transfer threshold* are used. These thresholds minimize the load balancing that is necessary and avoid excess distribution of jobs from one node to another.

Nodes with fewer jobs than the *minimum threshold* will initiate a load balancing request. The threshold should be a low value (1 or 2) and should never be greater than the current system load level. Therefore, in light load situations, unnecessary load balancing operations will not be triggered.

When the number of jobs queued at a given node exceeds the *maximum threshold* value, excess jobs are distributed to its neighbors. This parameter varies exponentially and is a function of current system load level and the network diameter. As network load increases, it is not as necessary to engage in load balancing activities because it is likely that all nodes have sufficient jobs to process.

The formula that is used to compute the maximum threshold is $M = S + 2^{S/D}$ where:

M = maximum threshold
S = present estimate of the system load value
D = number of communication stages in the network

If $S < D$, the maximum is less than D, and hence max = D.

The *transfer threshold* limits the maximum number of jobs that can be distributed at a time. The purpose is to reduce network traffic that can result from excessive job migration. During testing, it was found out that setting the *transfer threshold* to a small value does not degrade the effectiveness of the load balancing algorithm.

3.5 Information Transfer Policy

There are two cases in which a given node can indicate to a neighbor node a need for jobs. These cases are:

- When a load balancing request is routed to the next stage of nodes.
- When the updated system load level is distributed through the network.

In both of these cases, a node calculates its need for jobs by subtracting the number of jobs queued for processing from the current system load level. The transfer threshold effectively limits the effect of this calculation.

4 Implementation Details

4.1 Data Structures

The following data structures are required at each node in order to accomplish load balancing:

- The current estimate of the average number of jobs per node; the minimum, maximum, and transfer thresholds.
- A counter to track the number of load balancing requests that have passed this node. This count is increased when a load balancing request is received or initiated. It is decremented upon receipt of a distribution message that triggers the updating of current system load level.
- Because multiple load balancing requests can be processed simultaneously, arrays are used to to gather balancing and distribution messages. Note that the count of predecessor nodes corresponding to a given node is determined by the applying the *exor* operation to a given node's binary number and the binary number of the node that originated the load balancing request. Since in any communication path there are 2^{d-2} nodes in the network that must gather more than one message (d is the number of communication stages in the symmetric broadcast network), these arrays should be dimensioned at 2^{d-2}.

  During processing, to determine which array entry is to be used, a node applies the *exor* operation to the lower $d - 2$ bits of the node's binary number with the binary number of the node that originated the load balancing request.

The first of the three arrays is used to count the number of load balance messages gathered from predecessor nodes for a particular communication path from an originating node. The second array, similarly, counts the number of distribution messages gathered before updating the system load level. The final array is used to accumulate the estimate of the total system load as balance messages are received from predecessor nodes.

4.2 Initiating Load Balancing Requests

Load balancing requests are initiated when the number of jobs queued for processing at a given node fall below the minimum threshold. Load balancing can also occur if the node at the final stage of communication receives a distribution of jobs that cause the length of its queue to exceed the maximum threshold.

If there are no load balancing requests outstanding, the initiating node:

- Indicates that there are $\log P$ distribution messages to be gathered before this request is completed.
- Increments the count of load balancing messages that are being processed.
- Routes the load balancing request messages to all of its stage 1 neighbors. The local queue size is included in the message that is directed to the node's first successor. Also included in all balance messages is the node's need for jobs as described in Section 3.3.

If one or more load balancing requests are already outstanding, the balance initiating function exits with no action taken.

4.3 Receiving Load Balancing Requests

When a load balancing request is received, any needed jobs, if possible, are sent back to the predecessor node.

If this is the first balance message received from a given predecessor node:

- Counts of balance and distribution messages to be gathered are initialized.
- The number of balance requests that are being processed are incremented.

If more balance messages are still to be gathered:

- The total number of system jobs queued for processing is accumulated.
- The count of balance messages still to be received are updated.

If this is the last balance message to be received, the program decides whether this is the last stage of communication. If no, the balance messages are routed to the neighbor nodes at the next stage. The local queue size is included in the message that is directed to the node's first successor. Also included in all load balance messages is the node's need for jobs as described in Section 3.3.

If this is not the last stage of communication:

- The estimate of the system load level is adjusted.
- The count of balancing requests being processed is decremented.
- The load update job distribution messages (including any need for jobs) are routed to all of the node's first stage neighbors.

4.4 Initiating Job Distributions

If balancing requests are being processed, no action is taken.

Otherwise, if the queued number of jobs to process exceed the maximum threshold, a distribution message is sent to all stage 1 neighbors. The number of jobs distributed is computed by subtracting the system load level from the size of the local job queue. Note that the transfer threshold limits the maximum number of jobs that can be distributed.

4.5 Receiving Job Distributions

When a job distribution is received by a node:

- All jobs received are added to the local job queue.
- If the predecessor node is lightly loaded, return the needed jobs if possible.
- If the received distribution message requires a system load update and all predecessor distribution messages have been gathered, (i) update system load and threshold values, (ii) decrement the count of balance requests being processed, and (iii) route the distribution message to the next stage neighbors.
- If this is the last stage of communication and there is still an excess of messages, another load balancing request is triggered. Otherwise, route the excess to the nodes at the next stage.

5 Testing Procedures

5.1 Simulation Environment

The PVM (Parallel Virtual Machine) is used to create a parallel environment with up to 32 processes. A simulation program spawns the appropriate number of child processes to create the hypercube environment. The list of all process identifiers and an initial load of jobs are routed through the hypercube to all of the remote nodes. Each node then begins to processes the initial job list that it receives and makes use of the load balancing algorithm to even the work load among the other nodes of the network.

In addition to the initial load, each node generates additional jobs to be processed. Namely, ten job creation cycles are processed. Job generation at each node follows a Poisson distribution. During each of the job creation cycles the number of jobs that are generated at each node are varied. This is accomplished by randomly picking different λ constants which control how many jobs are generated. Therefore, both heavy and light system load condition are simulated.

Jobs are processed by delaying for the designated time period. Each process has a simulation clock

that accurately measures job processing and idle time.

When all of the jobs that were generated (initially and during the 10 job creation cycles) have been processed, the program terminates.

The programs can be given differing run time parameters to vary the size of the initial job load, the length of each job creation cycle, the average job length, and the average number of jobs to be generated per job creation cycle. In addition, the initial load can be randomly distributed to all nodes (*normal load*); or it can be distributed to a small subset of nodes (*extreme load*).

Three test runs have been prepared:

Figure 4 - Ten jobs per node are randomly distributed throughout the network as an initial load. The jobs generated during execution are more than can be processed by the network. Job duration averages one second. This test simulates a *Heavy System Load.*

Figure 5 - Fifty jobs are distributed to a small subset of nodes as an initial load. A light load of jobs is generated as the load balancing algorithm is processed. Job duration averages two seconds. This test simulates the *Transition from Heavy to Light System Load.* The load balancing algorithm also needs to adjust for the initial imbalance in the load at each node.

Figure 6 - A small number of jobs are initially distributed to a small subset of nodes. A light load of jobs are created as the algorithm executes. This test simulates a *Light System Load.*

5.2 Alternate Load Balancing Algorithms

Analogous programs have been implemented to compare the performance of the Symmetric Broadcast algorithm to other popular algorithms. Random, Gradient, Sender Initiated, Receiver Initiated, Adaptive Contracting, and no balancing programs have been tested in the same way as described above.

6 Summary of Experimental Results

6.1 Performance Metrics

The data and line charts that are included in Figures 4, 5, and 6 measure the comparative performance of the various balancing algorithms. The data records the statistical counts and totals that were obtained during execution of the simulation.

The X axis of the line charts show the number of nodes that were tested. The Y axis tracks variables that are useful to measure.

The following charts are included for each test:

(a) **Message Traffic Comparison by Node** which measures the total number of load balancing messages that were sent by any of the nodes.

(b) **Maximum Variance in Node Processing** which measures the difference in processing time between the most busy node of the network and the least busy node.

(c) **Total Time to Compute** which measures the time it took for the longest running node to complete its processing.

(d) **Maximum Variance by Node in Idle Time** which measures the time difference between the most idle and the least idle node.

(e) **Total Jobs Transferred** which measures the total number of job transfers from one node to another that occurred.

6.2 Summary

As expected, the program with *no load balancing* performs by far the worst. The *Random* algorithm, although providing significant improvement, nevertheless is less effective than the remaining algorithms.

The *Sender Initiated* and *Receiver Initiated* algorithms both perform better than the *Random* algorithm. In light to moderate loads, the receive algorithm has the disadvantage of generating more network traffic. Both of these approaches have inherent deficiencies.

Receiver Initiated algorithms, for example, tend to result in excessive load balancing activity at low system loads. this is because all nodes are polling neighbor nodes to find jobs that they can process. To overcome this deficiency, a time delay of one second has been introduced after a polling operation has been completed. This delay can unnecessarily increase the idle time, however.

Sender Initiated algorithms, on the other hand, can cause job thrashing to occur at high system loads. This has been overcome by reducing the number of job transfers that are done at high load levels. In addition, by limiting the migration of jobs, the potential for jobs to be routed back and forth is greatly reduced. The down side of these refinements, is that even in high load situations, it is possible for one or more nodes to be lightly loaded. Limiting job migration can sometimes prevent a "good" transfer decision to be made to make up for prior "poor" transfers.

The *Gradient* algorithm balances the load quite well. It has none of the above deficiencies. Unfortunately, lightly loaded nodes can sometimes receive too many messages from the overloaded nodes. This often results in excessive network traffic.

The *Adaptive Contracting(acwn)* and the *Symmetric Broadcast(symmetric)* algorithms were able to more evenly balance the generated system load than any of the other algorithms that were described. Of the two, the *Symmetric* method achieved the best results.

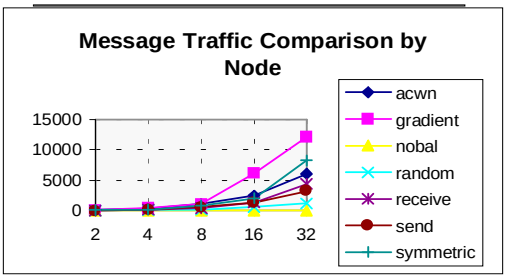
However, this improvement is achieved at the expense of more job transfers. During testing, it was apparent that the *Symmetric* algorithm is very sensitive to the threshold values that are chosen. It is desirable to choose these values in such a way as to minimize the processing of load balancing requests, since the associated processing is "expensive".

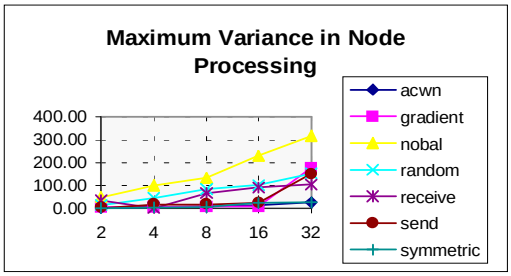
7 Conclusions and Future Research

The algorithm using symmetric broadcast networks has been proven to be an excellent approach to load balancing. The empirical results show that this approach can be used to very effectively balance

nodes	acwn	gradient	nobal	random	receive	send	sbc
2	109	57	0	15	13	35	46
4	299	460	0	76	165	156	130
8	1075	1079	0	183	433	558	836
16	2460	6118	0	557	1380	1297	2012
32	6026	12149	0	1177	4382	3207	8233

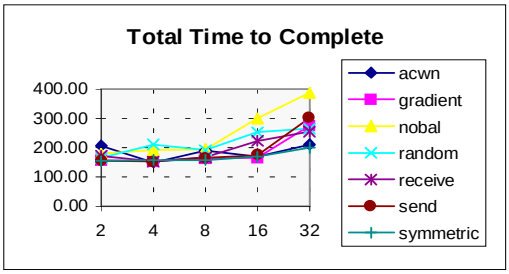
nodes	acwn	gradient	nobal	random	receive	send	sbc
2	0.10	4.10	47.70	13.20	34.40	0.10	0.50
4	3.90	3.50	98.40	45.20	0.80	15.10	1.60
8	6.50	7.60	133.40	82.90	65.10	16.80	4.10
16	10.20	5.90	229.40	101.49	89.70	22.10	22.50
32	24.80	177.40	317.20	153.50	104.30	151.20	25.60

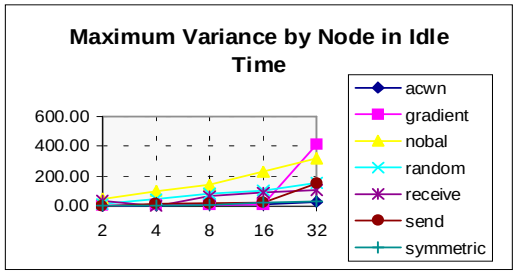




nodes	acwn	gradient	nobal	random	receive	send	sbc
2	207.40	157.00	178.80	165.50	172.10	155.00	155.20
4	148.50	153.50	191.60	210.60	152.00	152.80	158.00
8	190.20	162.90	194.10	191.10	166.20	165.40	158.10
16	168.20	165.70	300.00	253.40	223.20	174.50	168.90
32	210.20	274.70	387.40	264.30	253.70	301.70	200.50

nodes	acwn	gradient	nobal	random	receive	send	sbc
2	0.10	4.20	47.70	13.20	34.40	0.28	1.96
4	3.98	5.27	98.49	45.46	1.32	15.12	2.27
8	6.68	8.13	140.52	83.72	65.23	17.25	6.29
16	10.63	11.83	230.32	102.28	90.00	23.24	24.20
32	27.85	411.03	317.69	153.94	107.59	152.52	31.05





nodes	acwn	gradient	nobal	random	receive	send	sbc
2	190	28	0	157	6	130	23
4	387	202	0	467	75	347	241
8	1480	1009	0	863	160	878	1132
16	3436	11348	0	2561	464	1895	5415
32	9585	8908	0	4278	1257	5477	13766

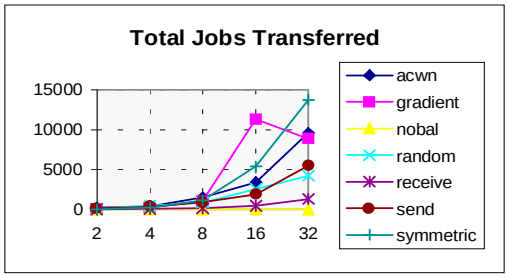
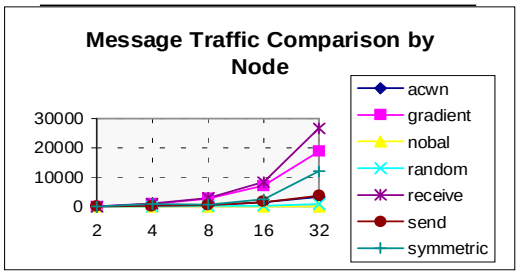
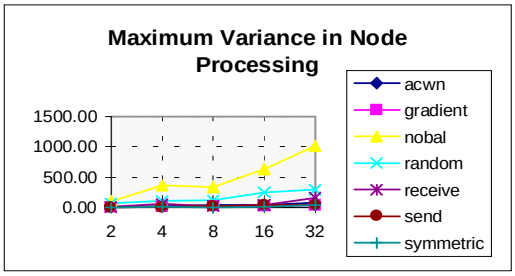


Figure 4: Heavy System Load

nodes	acwn	gradient	nobal	random	receive	send	sbc
2	60	43	0	13	53	28	66
4	196	902	0	66	1124	182	928
8	562	2809	0	147	2966	542	732
16	1386	7237	0	337	8445	1500	2428
32	3306	18898	0	786	26641	3583	12013

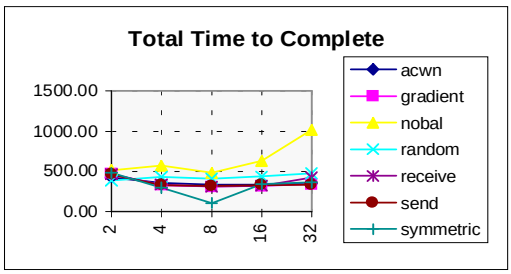
nodes	acwn	gradient	nobal	random	receive	send	sbc
2.00	11.30	8.70	102.30	69.30	4.90	3.70	1.90
4.00	21.60	13.20	360.20	110.10	59.00	5.10	16.90
8.00	44.10	23.00	335.20	122.10	22.90	27.80	2.10
16.00	46.10	22.60	630.50	251.50	45.10	42.00	19.80
32.00	83.30	39.60	1015.00	291.20	158.40	42.50	44.40

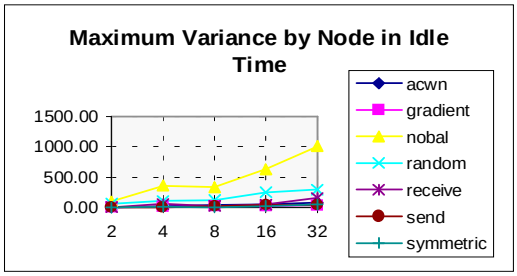




nodes	acwn	gradient	nobal	random	receive	send	sbc
2	428.80	469.50	516.30	389.00	462.70	467.00	484.30
4	357.30	332.50	569.70	429.70	339.90	328.30	300.60
8	336.20	311.40	480.40	415.10	311.30	318.40	109.30
16	335.10	314.70	630.60	440.90	324.10	327.80	343.80
32	346.90	340.80	1015.10	477.20	424.90	339.50	362.00

nodes	acwn	gradient	nobal	random	receive	send	sbc
2	11.32	8.70	102.34	69.30	4.90	3.70	1.90
4	21.77	13.20	360.20	110.23	59.25	6.70	17.74
8	44.16	24.15	335.46	122.51	22.97	29.12	5.56
16	47.53	23.78	631.45	251.50	46.61	43.21	21.26
32	84.75	41.71	1015.34	291.22	158.78	44.27	46.46





nodes	acwn	gradient	nobal	random	receive	send	sbc
2	144	28	0	180	24	209	32
4	525	293	0	495	190	390	553
8	1615	838	0	1002	404	786	600
16	3439	6750	0	1996	1325	1849	4077
32	8444	16416	0	4083	3761	3833	15947

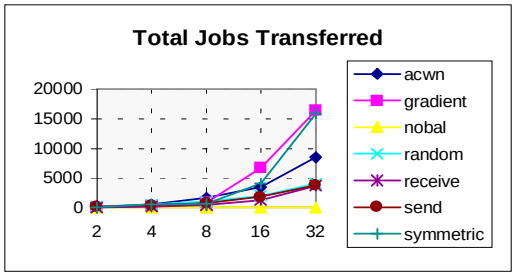
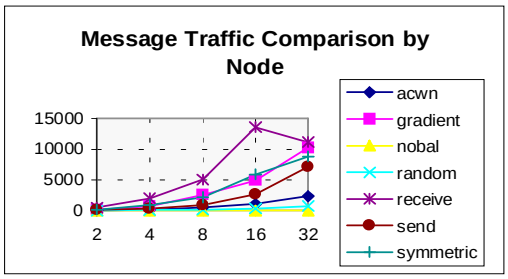
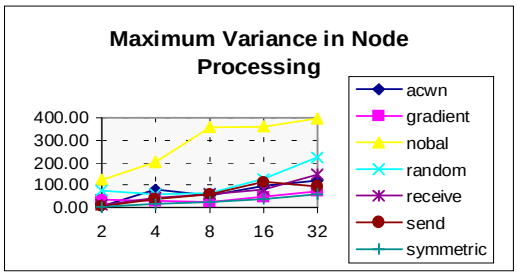


Figure 5: Transition from Heavy to Light System Load

nodes	acwn	gradient	nobal	random	receive	send	sbc
2	41	167	0	15	482	76	133
4	193	738	0	53	2008	303	925
8	486	2609	0	139	5091	933	2206
16	1099	4931	0	316	13593	2626	5912
32	2334	10224	0	786	11177	7202	8810

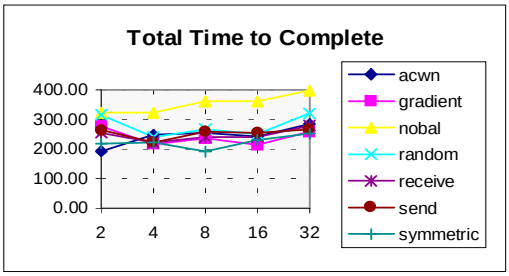
nodes	acwn	gradient	nobal	random	receive	send	sbc
2	5.80	35.10	124.30	76.00	11.70	7.10	1.40
4	78.90	29.30	204.20	60.00	43.10	39.70	16.00
8	51.80	25.20	358.90	63.00	59.80	60.10	25.80
16	95.10	47.80	360.40	127.70	80.50	114.20	40.00
32	121.60	73.90	397.10	224.10	148.80	92.80	59.60

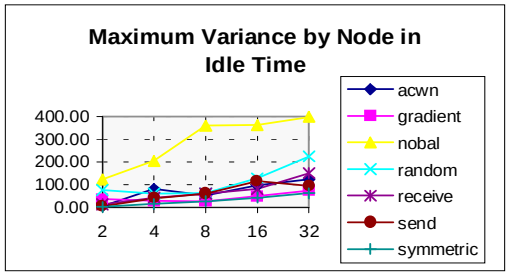




nodes	acwn	gradient	nobal	random	receive	send	sbc
2	192.70	277.40	322.00	315.70	254.00	263.40	218.70
4	248.10	215.70	321.50	241.10	221.00	221.80	221.60
8	254.80	236.00	360.50	267.00	239.80	257.50	191.20
16	242.10	212.50	360.50	248.50	241.00	255.00	230.00
32	283.20	258.50	297.20	320.20	278.40	267.40	251.50

nodes	acwn	gradient	nobal	random	receive	send	sbc
2	7.16	35.10	124.30	76.70	13.18	7.82	1.40
4	79.94	30.01	205.61	60.56	43.79	40.33	16.56
8	53.03	26.85	359.75	63.44	60.69	61.03	27.30
16	95.94	48.72	361.84	128.98	81.41	116.39	41.66
32	123.23	74.84	398.42	224.82	151.17	93.66	61.88





nodes	acwn	gradient	nobal	random	receive	send	sbc
2	96	93	0	136	91	119	245
4	306	287	0	297	164	248	1214
8	903	950	0	674	456	638	2133
16	1492	2015	0	1184	1013	1130	5485
32	3356	3600	0	2968	1522	2772	9521

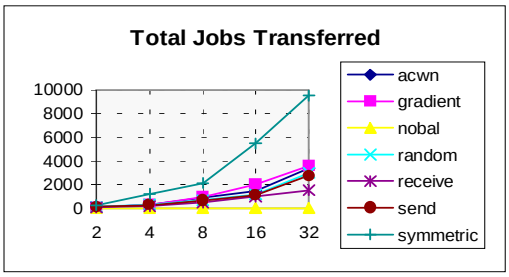


Figure 6: Light System Load

network load and is, in fact, superior to the other algorithms analyzed.

As we discussed in section 6, the choice of thresholds is critical to achieving effective load balancing while minimizing the network traffic required to achieve these results. A scientific study that determines how these thresholds should be set is an important area for consideration. In addition, further adaptations of the symmetric broadcast approach for use on other network topologies should be analyzed. Generalizations of the algorithm for use on a wide variety of multicomputer configurations would greatly broaden its possible applications.

Another area for research is to analyze the effect of altering the the definition of "system load". In this paper, local queue size determines system load. However, other parameters such as job length, communication cost, and execution dependencies could alter how balancing should be accomplished.

References

[1] S.K. Das and S.K. Prasad, "Implementing Task Ready Queues in a Multiprocessing Environment", *Proceedings of the International Conference on Parallel Computing*, Pline, India, pp. 132-140, Dec 1990

[2] S.K. Das, C-Q Yang, and N.K. Leung, "Implementation of Load Balancing in Multiprocessor Systems Using a Symmetric Broadcast Network", *Proceedings of the 1992 International Conference of Parallel and Distributed Systems*, Hsinchu Taiwan, pp. 589-596, 1992.

[3] S.K. Das, S.K. Prasad, C.Q. Yang, and N.M.Leung, "Symmetric Broadcast Networks for Implementing Global Task Queues and Load Balancing in a Multiprocessor environment", *Technical Report*, CRPDC-92-1, Department of Computer Science, University of North Texas, 1992.

[4] N.G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems", *Computer*, pp. 33-44, December 1992.

[5] D.Eager, G. Lazowska, and J. Zahorjan. "Adaptive Load Sharing in Homogeneous Distributed Systems", *IEEE Transactions on Software Engineering*, Vol SE-12, No. 5, pp. 662-675, 1986.

[6] F.C.H Lin and R.M. Keller, "The Gradient Model Load Balancing Method", *IEEE Transactions on Software Engineering*, SE-13, pp. 32-38, 1987.

[7] R. Luling, B. Monien, and F. Ramme, "Load Balancing in Large Networks, A Comparative Study", *Proceedings of the Third Symposium on Parallel and Distributed Processing*, Dallas TX, pp. 686-689, 1991.

[8] D.L. Eager et al., "A comparison of Receiver Initiated and Sender Initiated Adaptive Load Sharing", *Performance Evaluation*, Vol. 6, pp. 63-68, 1986.

[9] C.G. Rommel, "The Probability of Load Balancing Success in a Homogeneous Network", *IEEE Transactions on Software Engineering*, pp. 922-923, Sept. 1992.

[10] M.D. Feng and C.K. Yuen, "Dynamic Load Balancing on a Distributed System", *Proceedings of the Sixth IEEE Symposium on Parallel and Distributed Processing*, Dallas, TX, pp. 318-325, Oct 1994

[11] L.V. Kale, "Comparing the Performance of Two Dynamic Load Distribution Methods", *Proceedings International Conference on Parallel Processors*, Vol I, pp. 8-12, 1988.

[12] S. Pulidas, D. Towsley, J. A. Stankovic, "Embedding Gradient Estimators in Load Balancing Algorithms", *Proceedings IEEE 8th International Conference on Distributed Computing Systems*, pp. 482-490, 1988.

[13] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems", *Proceedings ACM Computer Network Performance Symposium*, pp. 47-55, April 1982.

[14] B.A. Shirazi, A.R. Hurson, K.M. Kavi, "Scheduling and Load Balancing in Parallel and Distributed Systems", *IEEE Computer Society Press*, Los Alamitos, CA 90720-1264, 1995.

[15] B.A. Shirazi, M. Wang, G. Pathak, "Analysis and Evaluation of Heuristic Methods for Static Task Scheduling", *Scheduling and Load Balancing in Parallel and Distributed Systems*, pp. 50-60, IEEE Computer Society Press, Los Alamitos, CA 90720-1264, 1995.

[16] V. Sarkar and J. Hennessy, "Compile-time Partitioning and Scheduling of Parallel Programs", *Scheduling and Load Balancing in Parallel and Distributed Systems*, pp. 61-70, IEEE Computer Society Press, Los Alamitos, CA 90720-1264, 1995.

[17] B.A. Shirazi et al. "Analysis and Evaluation of Heuristic Methods for Static Task Scheduling", *Scheduling and Load Balancing in Parallel and Distributed Systems*, pp. 96-111, IEEE Computer Society Press, Los Alamitos, CA 90720-1264, 1995.

[18] M.R. Eskicioglu, "Design Issues of Process Migration Facilities in Distributed Systems", *Scheduling and Load Balancing in Parallel and Distributed Systems*, pp. 414-424, IEEE Computer Society Press, Los Alamitos, CA 90720-1264, 1995.

[19] G. Cybenko, "Dynamic Load Balancing for Distributed Memory Multiprocessor", *Parallel and Distributed Computing*, pp. 279-301, Vol. 7, No. 2, Oct. 1989.

[20] G. Fox, A. Kalawa, and R. Williams, "The Implementation of a Dynamic Load Balancer", *Proceedings on Conference of Hypercube Multiprocessors*, pp. 114-121, 1987.

[21] K.G. Shin, Y.C. Chang, "Load Sharing in Hypercube Multicomputers for Real-Time Applications", *Proceedings 4th Conference Hypercubes, Concurrent Computers, and Applications*, pp. 617-621, Vol. 1, 1989.

[22] P. Krueger and M. Livny, "The Diverse Objectives of Distributed Scheduling Policies", *Proceedings IEEE 7th International Conference on Distributed Computing*, Systems, pp. 242-249, 1987.