Static Load Distribution for Communication Intensive Parallel Computing in Multiclusters

Eric M. Heien, Noriyuki Fujimoto, Kenichi Hagihara
Graduate School of Information Science and Technology, Osaka University
Toyonaka, Osaka 560-8531, Japan
{e-heien, fujimoto, hagihara}@ist.osaka-u.ac.jp

Abstract

In this paper, we examine load distributions to minimize total run time in multi-cluster parallel computing algorithms by applying divisible load theory techniques. Even with homogeneous processor speeds, parallel computations in multi-clusters that evenly assign load can run at less than maximum efficiency due to communication heterogeneity. Using a modified version of the LogP parallel computing model, we propose a general technique of assigning load among multiple clusters to minimize the time each processor spends waiting. This technique is used to determine optimal load distribution for spin glass simulation and parallel bucket sort in multi-cluster systems. It also allows fast analysis of the effects of adding processors or clusters to the computation. We experimentally demonstrate the accuracy of our model, and show how it eliminates wait time in multi-cluster parallel computations. Using load distributions derived from our technique results in an execution time decrease of up to 50%, depending on the degree of heterogeneity among clusters and communication characteristics of the computation.

1. Introduction

In recent years, cluster computing and computing grids have become more common. Combining multiple clusters into a single system is known as "multi-cluster computing". These clusters are often geographically separated, resulting in communication between clusters being slower than communication within a cluster. In a parallel computation on a multi-cluster setup, some communication may occur between processors inside a cluster and some communication may occur over the network connecting the clusters. Processors in the same cluster have relatively high bandwidth and low latency between each other compared to processors in different clusters. Therefore, one way to improve execution time of a communication intensive parallel computation is to take advantage of the faster intra-cluster communication. Also, in a multi-cluster computation it is difficult to estimate the optimal number of processors. The decrease in calculation time from using an additional cluster may be offset by an increase in overall communication time.

We now present an example of how load distribution can affect execution time in a multi-cluster. Suppose a computation consists of many steps, each step involving calculation followed by communication. The calculation and communication in a step is considered to be arbitrarily divisible, though one or both may depend on the task load assigned to a processor. For a given processor, step t must finish before step $t+1$ may begin. During each step processor P_i performs calculation, then communicates to P_{i+1}. Before P_i begins the next step it must receive data from P_{i-1}. In this example and in the rest of the paper, processor speeds are considered homogeneous. Generally for homogeneous processors, the task load assigned to each is equal (denoted the "EVEN" load distribution). In Figure 1, processors P_0, P_1, P_2 and P_3 are all in the same cluster connected by a fast link. P_0 communicates to P_1 over the fast link, as does P_1 to P_2 and P_2 to P_3. The rest of the processors communicate via a slow link. After P_0 finishes the first step, it must wait for communication from P_5 before continuing. This causes a delay for P_1 in the second step, which in turn delays P_2 in the third step and so on. This demonstrates that EVEN load distribution among homogeneous processors may not be optimal if communication speeds are heterogeneous.

If calculation and/or communication cost is in terms of task load, and the load may be arbitrarily divided among processors, the total run time of the computation can be improved in the following manner: processors with low communication costs (fast links) are given a larger proportion of the load, and processors with high communication costs (slow links) are given less. This is called the BIASED distribution. The effect of using the BIASED distribution can be seen in Figure 2, where processors P_0, P_1, P_2 are given

0-7695-3089-3/08 $25.00 © 2008 IEEE
DOI 10.1109/PDP.2008.58

IEEE computer society

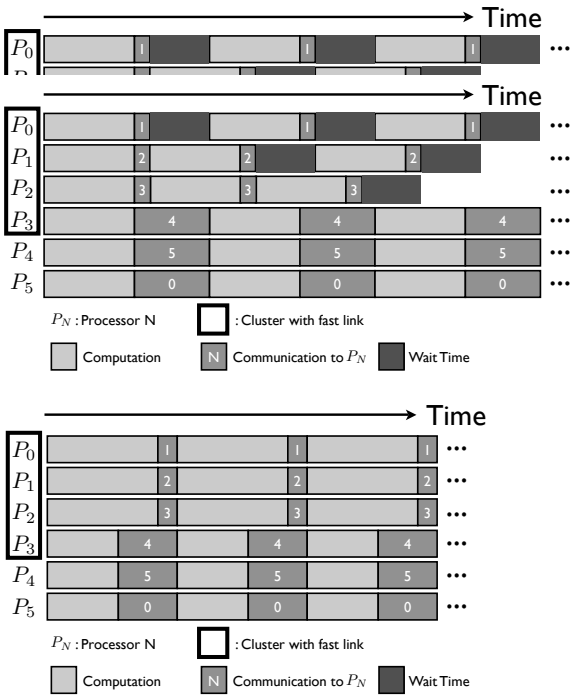

Figure 2. BIASED Load Distribution

more load and total time decreases. Note that even though P_3 is within the cluster, it is given a smaller load because it communicates over a slow link. This example uses a ring style communication pattern, however, the BIASED distribution is also effective with other communication patterns.

Derivation of BIASED load distributions has been investigated in [11]. However, in this work only a one-way ring communication pattern was examined, whereas we investigate a two-way ring as well as an all-to-all communication pattern. Previous work also assumed that calculation time is linear in terms of load, and communication time is constant, which does not hold for many parallel algorithms. In this work, we deal with two well known parallel algorithms - bucket sort and the Ising spin glass simulation.

In this paper we present a method for deriving BIASED distributions for parallel computation in multi-clusters with varying distributions of homogeneous processors. We assume there are only two types of communication links - fast intra-cluster links and slow inter-cluster links. BIASED distributions can be calculated for environments with heterogeneous processors speeds and multiple link speeds. However, this results in large systems of equations, so for simplicity we restrict discussion to homogeneous processors and only two types of links. Given the network and processor speeds, as well as characteristics of the parallel computation, we determine a BIASED load distribution that results in faster run time than the EVEN distribution.

The remainder of the paper is organized as follows. In Section 2 we introduce an extended version of the LogP model which takes into account network heterogeneity. We use this model to analyze calculation and communication time for multi-cluster parallel bucket sort and Ising spin glass simulation in Section 3. In Section 4, we experimentally verify our model and load distribution algorithm. In Section 5 we review related work on the subject, and offer our conclusions in Section 6.

2. Machine and Load Models

The LogP model [6] is a general parallel machine model used in the design and analysis of parallel algorithms. The model contains four key parameters to describe a system:

L: the latency of communicating a one word message from a source to a target.

o: the overhead time a processor spends sending or receiving a message.

g: the communication gap, i.e. the minimum time between receiving or sending consecutive messages. Corresponds to the reciprocal of communication bandwidth.

P: the number of processors.

In this paper, the LogP model is extended to recognize the difference between fast intra-cluster communication and slow inter-cluster communication. Instead of the L, o and g parameters, L_f, o_f, g_f (for fast links) and L_s, o_s, g_s (for slow links) are used. The parameter P remains unchanged. Using the extended model, BIASED load distribution for multi-cluster parallel computation can be determined. The extended model also allows fast investigation of the effects of adding or removing processors in a computation. In real systems there may be more types of links (intra-machine communication, network hierarchies, etc.), but here we restrict discussion to only two link types.

In this paper, we combine the extended LogP model with divisible load theory techniques to determine the fraction of input data or simulation space to be computed by each processor. Standard divisible load theory [2] assumes that the problem input data may be divided arbitrarily. Although this is an approximation of actual data (i.e. a bit cannot be divided), it is accurate for large sets of input data. Note that divisible load theory assumes no communication between processors during computation, and no calculation-communication dependence between processors. In contrast, our model assumes communication between processors, the cost of which is a function of the assigned processor load and communication pattern. We assume that task load is arbitrarily divisible among processors, with processor P_i receiving load f_i such that $\sum_0^{P-1} f_i = 1$. The EVEN distribution is defined as $f_0 = \ldots = f_{P-1} = \frac{1}{P}$. A load

distribution which minimizes wait time for processors in our extended LogP model is called a BIASED distribution.

3. Analysis of Algorithms

In this section, we determine BIASED load distributions for a 2D Ising spin glass simulation and parallel bucket sort algorithm by applying the extended LogP model and divisible load theory. These algorithms were chosen as representatives of two extremes in communication patterns - in the Ising spin model each processor communicates with 2 other processors in a ring pattern, whereas in parallel bucket sort each processor communicates with all other processors.

3.1. 2D Ising Spin Simulation

The Ising spin model is a theoretical model used to study the properties and behaviors of magnetic particles in a field [9]. The 2D Ising spin model represents a set of magnetic spins on a two dimensional toroidal square lattice of size S by S. Each spin is either +1 (up) or -1 (down), and spins interact with an energy of $-JS_{m,n}S_{x,y}$ where $S_{m,n}$ and $S_{x,y}$ are adjacent lattice points, and J is a positive real number representing the level of interaction. The Metropolis Monte Carlo [12] algorithm for spin systems is well known, and can be described as follows: 1) All lattice spins are randomly initialized to up or down. 2) A spin is selected and flipped if the energy level decreases. 3) If the energy level of the selected spin increases it is flipped with probability $P(\delta E) = e^{-\frac{\delta E}{K_b T}}$ where K_b is the Boltzman factor and T is the temperature. 4) Steps 2 and 3 are repeated until termination criteria are met.

Efficient parallel algorithms for the 2-dimensional Ising spin model have been studied in [13]. In our analysis, we use a variant of the "Truly Random" model described in the same work. The pseudocode for the truly random model simulation is shown in Algorithm 1.

In this analysis, the simulated lattice is divided in a striped layout such that each division has exactly 2 neighbors. Since the lattice is toroidal, the leftmost strip is neighbors with the rightmost strip. An example of both EVEN and BIASED load distributions for the Ising spin simulation is shown in Figure 3. This figure shows an EVEN distribution to the left, with $f_0 = f_1 = f_2 = f_3 = \frac{4}{16}$ and a BIASED distribution to the right, with $F_0 = \frac{2}{16}$, $F_1 = \frac{4}{16}$, $F_2 = \frac{6}{16}$. In the BIASED load distribution, processors are classified by the number of neighbors they communicate with via a fast link. A processor P_i that communicates with r neighbors over a fast link and $2 - r$ neighbors over a slow link is denoted as $P_{i,r}$ and the load fraction for this processor is F_r. This means that any two processors $P_{i,r}$ and $P_{n,r}$ have the same load F_r. In each step t, a processor performs

Algorithm 1 Truly Random 2D Ising Spin Simulation - One Monte Carlo Step

```
for P_0 do
    k ← 1, X ← []
    while Σ_{i=1}^{k} X_i < S^2 do
        X_k = random(1, 4S)
        k ← k + 1
    end while
    Broadcast X_1, ..., X_k to all processors
end for
for P_0, ..., P_{P-1} do
    t ← 1
    while t < k do
        Update X_t random spins on left half of strip, communicate updated border spins with neighbors
        Update X_{t+1} random spins on right half of strip, communicate updated border spins with neighbors
        t ← t + 2
    end while
end for
```

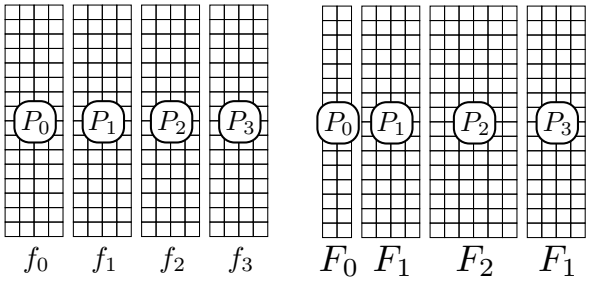

Figure 3. Example load distributions.

X_t spin changes on the left or right half of its strip (alternating every step), then communicates any spin changes on the edge of the strip to its left or right neighbor. The number of spin changes performed by $P_{i,r}$ is $F_r X_t$. X_t and X_{t+1} are uniformly random numbers from 1 to $4S$, so on average $P_{i,r}$ will perform $2F_r S$ spin changes per step. We denote the time to select and flip a spin as T_{spin}. Thus, the expected calculation time $T_r^{calc}(S)$ for two steps in processor $P_{i,r}$ is:

$$T_r^{calc}(S) = 4SF_r T_{spin} \tag{1}$$

After calculating the spin changes, each processor sends any changed boundary spin values on its left or right edges to the neighboring processors. There are an expected $2F_r S$ spin changes for one step, so the expected number of spin changes on one edge is $(2F_r S)/(S^2 F_r/2S) = 4$. The expected time $T_{fast}(S)$ (resp., $T_{slow}(S)$) for communication on a fast (resp., slow) link is denoted as:

$$T_{fast}(S) = L_f + 2o_f + 4g_f \tag{2}$$

$$T_{slow}(S) = L_s + 2o_s + 4g_s \quad (3)$$

The total time $T^r(S)$ per Monte Carlo step is:

$$T^r(S) = 4SF_rT_{spin} + (2-r)T_{slow} + rT_{fast} \quad (4)$$

The expected bandwidth usage is small enough that we ignore link contention between multiple processors. The number of processors with r fast links ($r \in [0,1,2]$) is denoted as C_r such that $\sum C_r = P$. For the EVEN distribution, $f_i = 1/P$ and the total time for one step is limited by the slowest link. In the case that there is a processor communicating only with slow links, the total time is:

$$T^0_{EVEN}(S) = \frac{4ST_{spin}}{P} + 2T_{slow}(S) \quad (5)$$

Wait time can be eliminated by setting the total times of each processor equal to each other ($T^0(S) = T^1(S) = T^2(S)$). This results in three linear equations (Equations 6, 7, 8) in three unknowns (F_0, F_1, F_2):

$$F_0C_0 + F_1C_1 + F_2C_2 = 1 \quad (6)$$

$$F_0 - F_1 = \frac{L_f - L_s + 2(o_f - o_s) + 4(g_f - g_s)}{4T_{spin}S} \quad (7)$$

$$F_0 - F_2 = \frac{L_f - L_s + 2(o_f - o_s) + 4(g_f - g_s)}{2T_{spin}S} \quad (8)$$

These linear equations are solved using Cramers rule to obtain the load distribution. The accuracy of these equations is demonstrated in Section 4.

3.2. Parallel Bucket Sort Analysis

Cluster based sorting has attracted attention in recent years as data collections start to span multiple machines and contain billions or trillions of objects [7]. A common parallel sorting algorithm is the parallel bucket sort. In this work, we use a variation on the parallel bucket sort, where the data is divided into chunks of objects to increase cache efficiency and avoid swapping. Pseudocode for the modified bucket sort is shown in Algorithm 2.

In the modified parallel bucket sort algorithm, there are S objects to be sorted which are divided among the processors $P_0, \ldots, P_{P-1}$. Each processor P_i holds a fraction f_i of the objects to be sorted, for a total of Sf_i objects. The objects on processor P_i are grouped into chunks. The chunk size is K_i objects, where $K_i = Kf_iP$ and K is the default chunk size. Therefore, there are $\lceil \frac{S}{PK} \rceil$ chunks on each processor. Each object is referenced by a 2 word key, and contains $(D-3)$ additional words of data. In this paper, data keys are assumed to be uniformly random in $[0, Q)$. Although actual data keys may have a non-uniform distribution, there are techniques (such as in sample sort [3]) to determine the key distribution and modify the algorithm appropriately. In step (1), shown in Figure 4, each processor gets a chunk and sorts the objects. We denote a processor P_i with r fast links and $P-r-1$ slow links as $P_{i,r}$ with load fraction F_r. The average sorting time for all the chunks at processor $P_{i,r}$ is:

$$T^r_{sort}(S) \approx \frac{SF_rT_{sort}}{K} \quad (9)$$

where T_{sort} is the time to sort a chunk of size K. In the parallel bucket sort algorithm, each processor is assigned a "bucket", or range of values. The bucket for processor P_i is $[\frac{Qi}{P}, \frac{Q(i+1)}{P})$. Any objects with keys in a processor's bucket range will be sent to that processor in step (2). Unlike in the Ising spin glass simulation, a processor does not need to receive all communications before processing another chunk. Steps (1) and (2) are repeated until the processor has completed sorting and sending all its chunks. However, before performing the final merge step (3) a processor must have received all objects from other processors. Therefore, a

Algorithm 2 Chunked Parallel Bucket Sort

Require: Object keys uniformly random in $[0, Q)$
1: **for** $P_0, \ldots, P_{P-1}$ **do**
2: **while** GetNextChunk() **do**
3: Sort chunk data keys with qsort()
4: **for** each $P_i \neq self$ **do**
5: Send objects in the sorted chunk with keys in $[\frac{Qi}{P}, \frac{Q(i+1)}{P})$ to P_i
6: **end for**
7: **end while**
8: Wait for other processors to finish sorting/sending chunks
9: Merge sorted sub-chunks
10: **end for**

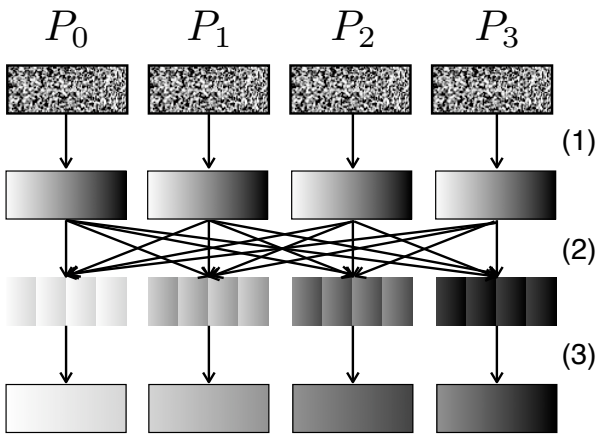

Figure 4. The parallel bucket sort.

slow processor can increase the total sort time by forcing other processors to wait before merging.

Since the keys are uniformly random, on average each processor will send KF_r objects of a chunk to any given processor. However, in this case we must take network contention into account. We assume that simultaneously sending M messages of size S on a link with gap g will result in total transmission time of MgS for each message. Therefore, the communication time on processor $P_{i,r}$ is:

$$T^r_{comm}(S) = \frac{S}{PK}(rT_f + (P - r - 1)T_s) \tag{10}$$

with $T_s = L_s + 2o_s + (P - r - 1)g_s DKF_r$ and $T_f = L_f + 2o_f + rg_f DKF_r$. This gives a total time of:

$$T^r(S) = T^r_{comm}(S) + T^r_{sort}(S) \tag{11}$$

As in the spin glass analysis, C_i indicates the number of processors with i fast links. The total load must equal one, $\sum_0^{P-1} C_i F_i = 1$. Processor finish times are set equal:

$$T^i(S) = T^{i+1}(S) \tag{12}$$

We assume that bandwidth costs are much greater than latency and overhead. For parallel bucket sort, this is true when $D \gg 1$ and/or $K \gg 1$. From Equation 12, we get:

$$F_{r+1}\delta_r = F_r \tag{13}$$

$$\delta_r = \frac{T_{sort} + \frac{DK}{P}(g_f r^2 - g_s(P - r - 1)^2)}{T_{sort} + \frac{DK}{P}(g_f(r + 1)^2 + g_s(P - r - 2)^2)} \tag{14}$$

Simplifying Equation 12 for $r = P - 1$ gives:

$$F_{P-1} = \frac{1}{\sum_{i=0}^{P-1}(C_i \prod_{j=i}^{P-2} \delta_j)} \tag{15}$$

We solve for F_{P-1} first, then work backwards to obtain the distributions for the other processors.

4. Experiments

To test the accuracy of the model and algorithms we performed several experiments. The experiments are intended to test the load distribution equations in actual programs. Appropriate sleep values were used in the program to simulate intra-cluster and inter-cluster latency, overhead and gap.

The cluster used in these experiments consists of 16 machines. Each machine has two 2.8 GHz Xeon processors, with 2 MB L2 cache and 2 GB RAM. The machines were connected by Gigabit ethernet over an HP Procurve switch. Network latency, overhead and gap were set at $L_f = 45\mu s$, $L_s = 400\mu s$, $o_f = 5\mu s$, $o_s = 10\mu s$, $g_f = 0.05\mu s$, $g_s = 0.5\mu s$. These values are based on measurements of the cluster Gigabit network and campus area network.

Table 1. Experiment cluster setups.

Setup	Total Procs.	Clusters × Processors
Spin 1	32	4×8
Spin 2	32	$2 \times 8, 4 \times 4$
Spin 3	32	$2 \times 8, 2 \times 4, 2 \times 2, 4 \times 1$
Spin 4	32	$2 \times 8, 16 \times 1$
Spin 5	32	$2 \times 8, 4 \times 2, 8 \times 1$
Spin 6	32	$8 \times 2, 16 \times 1$
Sort 1	16	$1 \times 8, 2 \times 4$
Sort 2	16	$1 \times 8, 4 \times 2$
Sort 3	32	$1 \times 16, 2 \times 8$
Sort 4	32	$1 \times 16, 1 \times 8, 2 \times 4$
Sort 5	32	$1 \times 16, 4 \times 4$
Sort 6	32	$1 \times 16, 8 \times 2$

The Ising spin simulation and parallel sort programs were written using pthreads. A main computation thread does the sorting/spin flipping and sending, while a receive thread handles incoming data. This allows processes to receive data while performing computation. Message passing between processes was implemented with MPICH2 v1.05p4. The multi-cluster setups for the experiments are shown in Table 1. These cluster sizes were chosen to represent a range of possible configurations. Each setup consists of a number of clusters, each containing 1, 2, 4, 8 or 16 processors. For example, cluster setup Sort 4 in Table 1 has 1 cluster of 16 processors, 1 cluster of 8 processors and 2 clusters of 4 processors.

4.1. Experimental Setup

Load distributions used in the Ising spin algorithm and parallel bucket sort experiments are shown in Table 4. These distributions were calculated using the equations in Section 3. As expected, a smaller load is allocated to processors with fewer fast links, even though processor speeds are homogeneous. For the Ising spin simulation a lattice size of $S = 8192$ was used, and a total of 20 Monte Carlo steps were performed. The average time to select and flip a spin was measured to be $T_{spin} = 0.550\mu s$. For the parallel bucket sort, the total number of objects was $S = 1 \times 10^7$ and chunk size was $K = 4096$ objects. For this chunk size, the average sort time was $T_{sort} = 3.42ms$. In both experiments, the programs were executed 10 times for each setup. A small experiment related to cluster selection was also performed. The parameters and results for this experiment are described in Section 4.4.

The results of experiments for the spin glass simulation show that using the BIASED load distribution significantly reduces total run time. A breakdown of run time for setup 4 can be seen in Figure 5. The effects of the EVEN load distribution are on the left half of the figure, showing significant wait time for $P_{i,1}$ and $P_{i,2}$. This is because processors

Table 2. Spin simulation load distributions.

Setup	EVEN	BIASED			
		F_0	F_1	F_2	
Spin 1	.0313	-	.0160	.0363	-
Spin 2	.0313	-	.0185	.0389	-
Spin 3	.0313	.0033	.0236	.0440	-
Spin 4	.0313	.0134	.0338	.0541	-
Spin 5	.0313	.0084	.0287	.0491	-
Spin 6	.0313	.0211	.0414	-	-
	EVEN	F_1	F_3	F_7	F_{15}
Sort 1	.0625	-	.0405	.0845	-
Sort 2	.0625	.0327	-	.0923	-
Sort 3	.0313	-	-	.0203	.0422
Sort 4	.0313	-	.0157	.0212	.0440
Sort 5	.0313	-	.0164	-	.0461
Sort 6	.0313	.0148	-	-	.0477

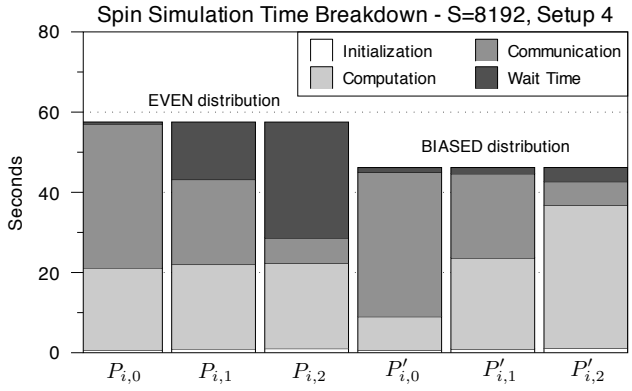

Figure 5. Spin simulation time breakdown.

with fast links complete communication before processors with slow links, which eventually causes fast link processors to wait for communication. The effects of the BIASED load distribution are shown on the right half, where wait time is nearly eliminated. The BIASED load distribution effectively increases the calculation load for $P'_{i,1}$ and $P'_{i,2}$ to compensate for the lower communication cost on these processors. All other setups showed similar results in run time from the BIASED load distribution. Due to the random nature of the algorithm, it is difficult to completely eliminate wait time. Also, cache effects not included in the model can affect total wait time. We found that the model became less accurate for large lattice sizes ($S > 2^{15}$) where each processors sub-lattice could not easily fit in cache.

4.2. Spin Glass Simulation Results

Figure 6 shows the comparison of the total times for EVEN and BIASED distributions. In all setups, the BIASED distribution outperformed the EVEN distribution.

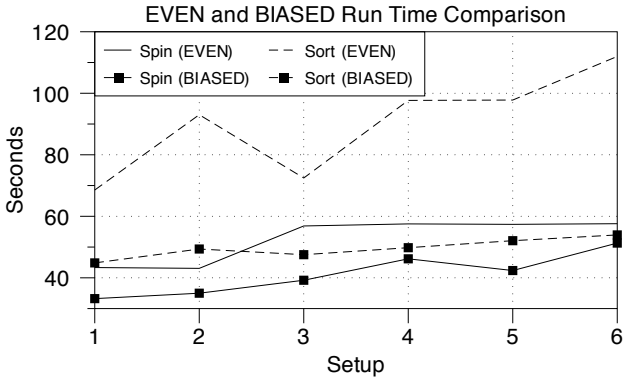

Figure 6. Run time comparison between EVEN/BIASED distributions.

This was particularly noticeable in setups with a wide distribution of cluster types (setups 3, 4, 5). This is because there is a wider range of communication speeds for these setups, and thus a greater wait time with the EVEN distribution.

The accuracy of the model is shown in Table 7. In all setups, the difference between the predicted and actual run time was less than 5% or less. It is worth noting that the time predicted by the model is always less than the actual time. We believe this is due to the random nature of the algorithm combined with caching effects.

4.3. Parallel Bucket Sort Results

The results of the parallel bucket sort experiments also demonstrate the effectiveness of the BIASED distribution. In the parallel bucket sort, run time is dominated by communication, as the time to send an object is significantly higher than the time to sort it. This is exacerbated by network contention among processors. The run time breakdown for setup 4 is shown in Figure 7, with the EVEN distribution processors on the left and the BIASED distribution processors on the right. In the EVEN distribution, processors in larger clusters ($P_{i,15}$ and $P_{i,7}$) are forced to wait for processors in smaller clusters before merging. With the BIASED distribution, total run time is lowered. All other setups showed similar results from the BIASED distribution.

Comparison of EVEN and BIASED run times for all setups is shown in Figure 6. Run time decrease between the EVEN and BIASED distributions is greater for the parallel bucket sort than for the spin simulation. This is because most of the run time is taken by communication, which can be spread more evenly among clusters by using the BIASED distribution. Network contention also results in better performance by the BIASED distribution - more load in large clusters results in less network contention for slow inter-cluster communication.

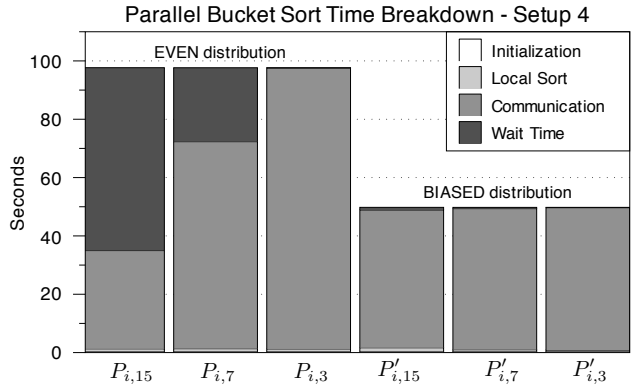

Figure 7. Parallel sort time breakdown.

Table 3. Run time comparison.

Setup	Model (secs)	Actual (secs)	Difference
Spin 1	31.3	33.2	5.7%
Spin 2	33.2	35.0	5.1%
Spin 3	37.0	39.2	5.6%
Spin 4	44.5	46.2	3.7%
Spin 5	40.7	42.4	4.0%
Spin 6	50.1	51.3	2.3%
Sort 1	45.8	44.8	2.2%
Sort 2	50.5	49.4	2.2%
Sort 3	47.6	47.5	0.2%
Sort 4	50.0	49.8	0.4%
Sort 5	52.3	52.1	0.4%
Sort 6	54.2	54.0	0.3%

It is worth noting that setups with more homogeneous clusters (Setups 1,3) show less improvement than setups with more cluster heterogeneity. For the parallel bucket sort on multiple identical clusters, the BIASED distribution is the same as the EVEN distribution and will be of no benefit. However, the BIASED distribution will be beneficial to the spin glass simulation in any environment with a cluster having 3 or more processors. This is due to the communication patterns of each algorithm. Table 7 shows the accuracy of the model. In all setups, the difference between the predicted and actual runtime was less than 6%.

4.4. Cluster Selection Analysis

In order to decrease run time, additional processors are often added to a computation. The models used to determine BIASED distributions can also be used to evaluate the effect of adding or removing processors/clusters in the computation. In this experiment, we demonstrate how the model is used to evaluate adding clusters.

In this experiment, the base system is 2 clusters of 8 processors (denoted "Base"). There is an option of adding one more 8-processor cluster (denoted "8P"), or 3 more 4-processor clusters (denoted "3x4P"). Adding 3x4P gives 50% more processing power than 8P, but also can result in more communication over the slow inter-cluster link. As shown in Table 8, the effect of adding 3x4P is nearly the same as adding 8P, both in the model and actual execution. For spin simulation, a larger lattice size S results in more calculation, but does not affect the communication time. Thus, 3x4P gives better performance as lattice size increases. The predicted and actual speed increase for the clusters closely match, demonstrating the effectiveness of using this model for determining processors to allocate to a parallel computation. By using the model, resource allocation decisions can quickly be made for a computation.

5. Related Work

Several works have studied job scheduling for co-allocation in multi-clusters [4] [10], though these focus on avoiding network congestion rather than balancing load. Much of the work in multi-cluster co-allocation assumes non-divisible tasks and constant communication costs, which may not be valid for many types of parallel computations.

Another method of minimizing parallel computation run time is dynamic load balancing [1] [8] [15] . This method balances processor load by transferring load from busy processors to less loaded processors. Although this technique is beneficial in an environment with dynamically varying resources, it does not allow analysis of the tradeoffs between different resources as our model does. Furthermore, dynamic load balancing is highly application dependent, and must be tailored for a given application and/or environment.

Improving execution time of sort on a heterogeneous cluster has been investigated [5], though this work focuses on clusters with heterogeneous processor speeds and homogeneous communication speeds. There has been some work in optimizing communication patterns in the Ising spin algorithm for hierarchical cluster setups [14], however, this work involves sublattice shaping using an EVEN distribution, and focuses on sweep selection rather than random selection. The work also assumes that the clusters are homogeneous, whereas our work deals with varying cluster sizes. Future work could combine sublattice shaping with BIASED load distribution for random selection.

6. Conclusion and Future Work

In this paper, we demonstrated how EVEN load distribution can cause wait time in multi-cluster parallel computation. We demonstrated a technique for determining BIASED load distribution in multi-clusters with homogeneous

Table 4. Cluster Addition Analysis.

Clusters	Lattice Size (S)	Model	Actual	Difference	Model Speedup	Actual Speedup
Base	4096	15.67	16.34	4.1%	n/a	n/a
Base + 8P	4096	11.83	12.85	7.9%	24.5%	21.4%
Base + 3x4P	4096	11.53	12.88	10.5%	26.4%	21.2%
Base	8192	54.42	53.22	2.3%	n/a	n/a
Base + 8P	8192	39.04	39.92	2.2%	28.3%	25.0%
Base + 3x4P	8192	36.25	37.68	3.8%	33.3%	29.2%
Base	16384	201.1	197.6	1.8%	n/a	n/a
Base + 8P	16384	139.6	142.1	1.8%	30.6%	28.1%
Base + 3x4P	16384	125.2	135.8	7.8%	37.7%	31.3%

nodes, and derived load distribution equations for a 2D Ising spin glass simulation and a parallel bucket sort. These equations were used to calculate BIASED load distributions for multi-cluster environments. Experiments demonstrated that the BIASED load distributions outperform the EVEN load distribution in all tested setups. We also demonstrated that these equations allow us to evaluate the effect of adding additional processors to a computation.

In future work, the model could be extended to include heterogeneous processor speeds or multiple link speeds. Further analysis of the model for other types of parallel computations would prove useful, though some parallel computations may not be treated as divisible and therefore could not be used with this technique. Finally, coupling this kind of divisible load analysis with a task scheduler would allow quick analysis of various resource allocation strategies and may allow for more efficient scheduling of parallel tasks in multi-cluster systems or grid environments.

Acknowledgments

This research was supported in part by Grant-in-Aid for Young Scientists (B)(18700058) and Grant-in-Aid for Scientific Research (B)(18300009) from the Japan Society for the Promotion of Science.

References

[1] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms. *Parallel and Distributed Systems, IEEE Transactions on*, 16(4):289–299, 2005.

[2] V. Bharadwaj, T. G. Robertazzi, and D. Ghose. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[3] G. E. Blelloch, C. E. Leiserson, B. M. Maggs, C. G. Plaxton, S. J. Smith, and M. Zagha. A comparison of sorting algorithms for the connection machine cm-2. In *Proceedings of the 3rd ACM symposium on Parallel algorithms and architectures*, pages 3–16, New York, NY, 1991. ACM Press.

[4] A. I. D. Bucur and D. H. J. Epema. Scheduling policies for processor coallocation in multicluster systems. *IEEE Trans. Parallel Distrib. Syst.*, 18(7):958–972, 2007.

[5] C. Cerin, M. Koskas, H. Fkaier, and M. Jemni. Improving parallel execution time of sorting on heterogeneous clusters. In *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing*, pages 180–187, Washington, DC, USA, 2004. IEEE Computer Society.

[6] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In M. Chen, editor, *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, San Diego, CA, May 1993. ACM Press.

[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[8] K. Devine, B. Hendrickson, E. Boman, M. S. John, and C. Vaughan. Design of dynamic load-balancing tools for parallel applications. In *ICS '00: Proceedings of the 14th international conference on Supercomputing*, pages 110–118, New York, NY, USA, 2000. ACM Press.

[9] E. Ising. *Beitrag zur Theorie des Ferromagnetismus*, pages 31, 253–258. Z. Physik, 1925.

[10] W. M. Jones, L. W. Pang, W. B. Ligon, and D. Stanzione. Bandwidth-aware co-allocating meta-schedulers for mini-grid architectures. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 45–54, Washington, DC, 2004. IEEE Computer Society.

[11] A. Legrand, H. Renard, Y. Robert, and F. Vivien. Mapping and load-balancing iterative computations. *IEEE Trans. Parallel Distrib. Syst.*, 15(6):546–558, 2004.

[12] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

[13] E. E. Santos, S. Feng, and J. M. Rickman. Efficient parallel algorithms for 2-dimensional ising spin models. In *Proc. 16th International Parallel and Distributed Processing Symposium*, page 135. IEEE Computer Society, 2002.

[14] E. E. Santos and G. Muthukrishnan. Efficient simulation based on sweep selection for 2-d and 3-d ising spin models on hierarchical clusters. *ipdps*, 14:229b, 2004.

[15] A. Y. Zomaya and Y.-H. Teh. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(9):899–911, 2001.