Reconfigurable Parallel Sorting and Load Balancing on a Beowulf Cluster: HeteroSort

Pamela Yang[1], Timothy M. Kunau[1], Bonnie Holte Bennett[1], Emmett Davis[1], Bill Wren [2]

[1] University of St. Thomas, Graduate Programs in Software, Mail # OSS 301, 2115 Summit Avenue, Saint Paul, MN 55105
PamelaY@uswest.att.net, kunau@ahc.umn.edu, bhbennett@stthomas.edu, EmmettDa@aol.com
[2] Honeywell Technology Center, 3660 Technology Drive, Minneapolis, MN 55418
Wren_Bill@htc.honeywell.com

Abstract. HeteroSort load balances and sorts within static or dynamic networks using a conceptual torus mesh. We ported HeteroSort to a 16-node Beowulf cluster with a central switch architecture. By capturing global system knowledge in overlapping microregions of nodes, HeteroSort is useful in data dependent applications such as data information fusion on distributed processors.

1 Introduction

Dynamic adaptability, both within an application's immediate distributed environment as well as future environments to which it will be ported, is a keystone feature for applications implemented on modern networks. Dynamic adaptability is a basis for fault tolerance. A system, which is dynamically adaptive, strives to withstand the assault of hardware glitches, electrical spikes, and component destruction. The research described in this paper set out to develop a high-speed load balancing algorithm, which would balance loads by sorting data across the network of nodes and resulted in developing a reconfigurable system for parallel sorting with dynamic adaptability.

1.1 Dynamic Adaptability

With the increased dependence on distributed and parallel processing to support general as well as safety-critical applications, we must have applications that are fault tolerant. Programs must be able to recognize that current resources are no longer available. Schedulers are employed in the presence of faults to manage resources against program needs using dynamic or fixed priority scheduling for timing correctness of critical application tasks.

We have taken a different approach and refocused on the design of elemental processes such as load balancing and sorting. Instead of depending on schedulers, we design process algorithms where global processes are completed using only local knowledge and recovery resources. This lessens the need for schedulers and eases their workload.[1]

1.2 Beowulf Clusters

Beowulf clusters are one of the most exciting implementations of Linux today. Originating from the Center of Excellence in Space Data and Information Sciences (CESDIS) at the NASA Goddard Space Center in Maryland, the project´s mission statement is:

> Beowulf is a project to produce the software for off-the-shelf clustered workstations based on commodity PC-class hardware, a high-bandwidth internal network and the Linux operating system.

The Beowulf project was conceived by Dr. Thomas Sterling, Chief Scientist, CESDIS. One of NASA´s imperatives has always been to share technology with universities and industries. With the Beowulf project, NASA has provided the Linux community with the opportunity to spread into scientific areas needing high performance computing power.[2]

1.3 Local Knowledge and Global Processes

An efficient network sort algorithm is highly desirable, but difficult. The problem is that it requires local operations with global knowledge. So, consider a group of data (for example, that of names in a phone directory) which is to be distributed across a number of processors (for example, 26). Then an efficient technique would be for each processor to take a portion of the unsorted data and send each datum to the processor upon which it eventually belongs (A's to processor 1, B's to processor 2, … Z's to processor 26).

A significant practical feature of HeteroSort is that in our experiments it load balances before it finishes sorting. Since HeteroSort detects when the system is sorted, it also detects termination of load balancing. Chengzhong Xu and Francis Lau in <u>Load Balancing in Parallel Computers: Theory and Practice</u> (Boston: Kluwer Academic Publishers, 1997) state:

---

[1] Examples of these fault tolerant efforts can be found in the work of Jay Strosnider and his colleagues at Department of Electrical and Computing Engineering, Carnegie Mellon University in the Fault-Tolerant Real Time Computing Project.
Katcher, Daniel I., Jay K. Strosnider, and Elizabeth A. Hinzelman-Fortino.
"Dynamic versus Fixed Priority Scheduling: A Case Study"
http://usa.ece.cmu.edu/Jteam/papers/abstracts/tse93.abs.html.

[2] For more information, see The University of St. Thomas Artificial Intelligence and High Performance Parallel Processing Research Laboratory's Beowulf cluster web page: Kunau, Timothy M. http://aibc.gps.stthomas.edu.

From a practical point of view, the detection of the global termination is by no means a trivial problem because there is a lack of consistent knowledge in every processor about the whole workload distribution as load balancing progresses.[4]

Thus the global knowledge that all names beginning with the same letter belong on a prespecified processor facilitates local operations in sending off each datum. The problem, however, is that this does not adequately balance the load on the system because there may be many A’s (Adams, Anderson, Andersen, Allen) and very few Q’s or X’s. So the optimal loading Aaa-Als on processor 1, Alb-Bix on processor 2, … Win-Zzz on processor 26) cannot be known until all the data is sorted. Global knowledge (the optimal loading) is unavailable to the local operations (where to send each datum) because it is not determined until all the local operations are finished. HeteroSort combines load balancing within sorting processes.

Traditionally, techniques such as hashing have been used to overcome the non-uniform distribution of data. However, parallel hash tables require expensive computational maintenance to upgrade each sort cycle, thus making them less efficient than HeteroSort, which requires no external tables.

1.4 Related Work

Much of the work in this area deals with linear arrays.[2,3] The general approach is to take linear sort techniques and use either a row major or a snake-like grid overlaid on a regular grid topology of processors.[1] The snake-like grid is used at times with a shear-sort or shuffle sorting program where there is first a row operation and then an alternating column operation. So, either the row or the column connections are ignored in each cycle.

2 Approach

HeteroSort is our load balancing and sorting algorithm. Our initial approach was to use four-connectedness (as an example of N-connectedness) for load balancing and sorting. In traditional linear sorts data is either high or low for the processor it is on, and is sent up or down the sort chain accordingly. Our approach differs in that we defined data to be very high, high, low, or very low. In order to do this we first defined a sort sequence across an array of processors as depicted in Figure 1.

Next we defined the four neighbors. This is easily understood by examining Node 7 in the example of sixteen processors shown in Figure 1. The neighbors for Node 7 are 2, 6, 8, and 10. When Node 7 receives its initial data, it sorts it and splits it into four quarters. The lowest quarter goes to Node 2, the next lowest quarter goes to Node 6, the third quarter goes to Node 8, and the highest quarter goes to Node 10. Thus, the extremely high and low data are shipped on “express pathways” across the coils of the snake network.

1 **2** 3 4

8	**7**	**6**	5
9	**10**	11	12
16	15	14	13

Fig. 1. The sort sequence is overlaid in a snake-like grid across the array of processors. The lowest valued items in the sort will eventually reside on processor 1 and the highest valued items on processor 16. Node 7's four connected trading partners are in bold: 2, 6, 8, and 10. When Node 7 receives its initial state, it sorts and splits the data into four quarters. The lowest quarter goes to Node 2. The next lowest quarter goes to Node 6, the third quarter to Node 8, and the highest quarter goes to node 10. Thus the extremely high and low data are shipped across the coils of the snake network.

The trading neighbors Node 2 and Node 10 which are not adjacent on the sort sequence (transcoil neighbors) provide a pathway for very low or very high data to pass across the coils of the snake network into another neighborhood of nodes. This provides an express pathway for extremely ill sorted data to move quickly across the network. The concept of four connectedness is easy to understand with an interior node like Node 7, but other remaining nodes in this example are edge nodes, and their implementation differs slightly.

Table 1. Trading partner list. Determining which data is kept at a node depends on how that node falls among the sort order of its neighbors. For example, node 1 falls below all of its neighbors and thus receives the lowest quarter.

Node	Odd Cycle	Even Cycle	Node	Odd Cycle	Even Cycle
1	**1** 2 4 8 16	**1** 16 4 8 2	9	8 **9** 10 12 16	8 **9** 16 12 10
2	1 **2** 3 7 15	1 **2** 15 7 3	10	7 9 **10** 11 15	9 7 **10** 15 11
3	2 **3** 4 6 14	2 **3** 14 6 4	11	6 10 **11** 12 14	10 6 **11** 14 12
4	1 3 **4** 5 13	3 1 **4** 13 5	12	5 9 11 **12** 13	11 9 5 **12** 13
5	4 **5** 6 8 12	4 **5** 12 8 6	13	4 12 **13** 14 16	12 4 **13** 16 14
6	3 5 **6** 7 11	5 3 **6** 11 7	14	3 11 13 **14** 15	13 11 3 **14** 15
7	2 6 **7** 8 10	6 2 **7** 10 8	15	2 10 14 **15** 16	14 10 2 **15** 16
8	1 5 7 **8** 9	7 5 1 **8** 9	16	1 9 13 15 **16**	15 9 13 1 **16**

Simply put, we use a torus for full connectivity. So nodes along the "north" edge of the array which have no north neighbors are connected (conceptually) to nodes along the "south" edge and vice versa (transedge neighbors). Similarly, a node along the "east" edge are given nodes along the "west" edge as east neighbors and so forth. The odd cycle column of Table 1 summarizes all the nodes of a sixteen node network.

Thus, the use of the torus for four-connectedness provides full connectivity. The result is a modified shear-sort where both row and column connections are used with each round of sorting. Furthermore, ill-sorted data is quickly moved across the network via torus connections.

The "express pathway" is a conceptual map of the sorting network. Ideally, the operating systems supports express pathways, such as in an Intel Paragon system where we first implemented our algorithm. Where this environmental support is missing, the cost of these non-adjacent operations is higher. In those environments where networks have edges, HeteroSort has three strategies. The first is to still implement the conceptual torus at the higher transmission cost. The second is to re-configure itself to the reality of some nodes having only two or three physical neighbors. A third strategy is particularly useful in heterogeneous environments, where we employ a genetic algorithm to determine the optimal network by minimizing transmission costs.

2.1 Beowulf Clusters

The major portion of this Beowulf background section is abstracted from CESDIS material on their web page: http://www.beowulf.org/

The Beowulf class of computers and its architecture are appropriate to the times. The increasing presence of computers in offices, homes, and schools, has led to an abundance of mass produced cost effective components. The COTS (Commodity Off The Shelf) industry now provides fully assembled subsystems (microprocessors, motherboards, disks and network interface cards). The pressure of the mass market place has driven the prices down and reliability up. In addition, shareware, freeware, and open source development; in particular, the Linux operating system, the GNU compilers and programming tools and the MPI and PVM message passing libraries, provide hardware independent software.

In the taxonomy of parallel computers, Beowulf clusters fall somewhere between MPP (Massively Parallel Processors, like the Convex SPP, Cray T3D, Cray T3E, CM5, etc.) and NOWs (Networks of Workstations). The Beowulf project benefits from developments in both these classes of architecture. MPP's are typically larger and have a lower latency interconnect network than Beowulf clusters. Most programmers develop their programs in message passing style. Such programs can be readily ported to Beowulf clusters. Programming a NOW is usually an attempt to harvest unused cycles on an already installed base of workstations in a lab or on a campus. Programming in this environment requires algorithms that are extremely tolerant of load balancing problems and large communication latency. These programs will directly run on a Beowulf.

A Beowulf class cluster computer differs from a Network of Workstations in that the nodes in the cluster are dedicated to the cluster. This eases load balancing. Also, this allows the Beowulf software provide a global process ID, enabling signals to be sent from one node to another node of the system..

The challenge for our HeteroSort has been to adapt a conceptual mesh torus to a Beowulf cluster architecture. A trade in benefits has been the increased expense of nearest neighbor transactions. In the Beowulf, all transactions pass through a switch. This expense trades for the benefit that all other transactions do not have to traverse a network, passing through intervening nodes.

2.2 Optimization of HeteroSort

HeteroSort's distributed approach can provide an efficient control mechanism for a wide variety of algorithms. It also provides "reconfiguration-on-fault" fault tolerance when a node or network error occurs. HeteroSort automatically reconfigures to account for the failed node(s), and the distributed data is not lost. However, efficient operation requires that major sort axis nodes should reside on near neighbor network physical processors. This minimizes communication costs for efficient operation. And, for a heterogeneous topology, or a homogeneous topology made irregular by failed nodes, automatically achieving this near neighbor configuration for the sort nodes is difficult.

Figure 2 indicates a homogeneous mesh made irregular by two failed nodes. The numbers in the boxes (nodes) indicate the node's position in the sort order.

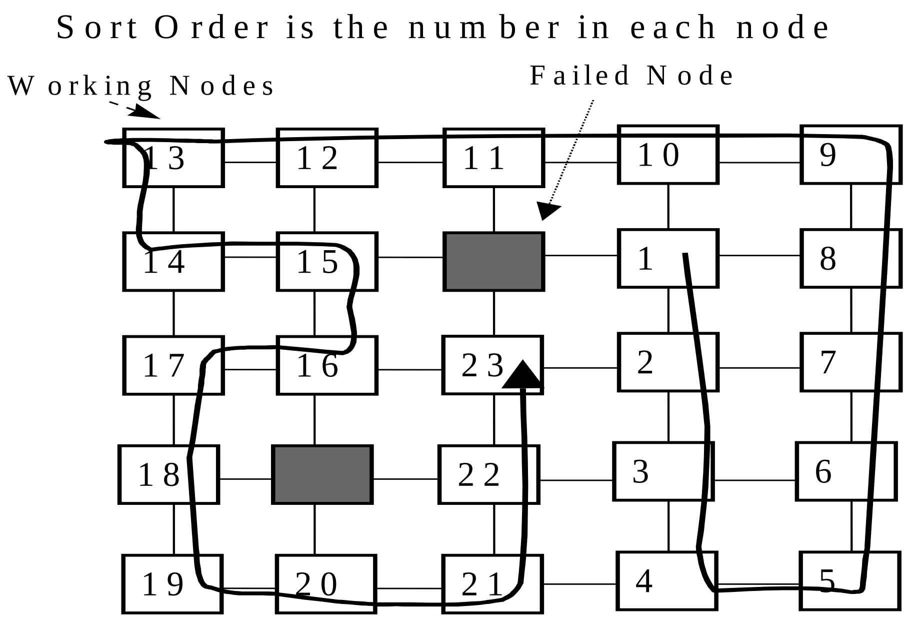

Fig. 2. Sort order is the number of each node. A homogeneous mesh of 25 nodes made irregular by two failed nodes requires a new sort order for efficient performance. The numbers in the boxes (nodes) indicate the node's position in the new sort order. The lowest valued items in the sort will eventually reside on processor 1 and the highest valued items on processor 23. This new order optimizes near neighbor relations.

We assume that a message cannot be sent across a failed node. To provide for online reconfiguration of the node sort order, we have developed an adaptive online sort order optimizer named the Scaleable Adaptive Load-balancing (SAL) Online Optimizer (SOO). SOO is performed by using a genetic algorithm which minimizes the total path length of the HeteroSort major sort axis, indicated on the figure by the line from one to 23. Note that other possible minimum path sort orders exist. Also note that for some topologies or failure patterns, strict near neighborness is not achievable. For these cases SOO defines the minimum path that includes store-and-forwards or traversals across other nodes. SOO can optimize given any combination of failed nodes and busses.

3 Fault Tolerance

The most important aspect of our algorithm is that it does not depend on a regular network topology (as, for example, a traditional shear sort does) because the torus can be superimposed on any physical architecture. This yields fault tolerance because our system can dynamically reconfigure itself, and easily accommodates "holes" in the connection. All that is required is for HeteroSort to change the partitioning schema in the data, and to stop sending data to a node when it is removed.

Three other aspects of fault tolerance result from this algorithm. First, since only local knowledge is used in the sort, the system is fault tolerant because it does not require global knowledge. Thus, individual nodes continue to operate regardless of the performance (or even existence) of other non-neighbor nodes.

Second, since each node keeps a backup copy of the data it sends off to its neighbors, if a node is eliminated during operation of the load balancing and sorting, its neighbors can make up for the loss of data. Third, the natural load balancing of the data during operation of the sorts adds a degree of fault tolerance. With data evenly distributed across nodes, then the loss of a node means the minimal loss of data to the system. The intent is to build minimum weight spanning trees and to use them in improving sort efficiency.

5.1 Future Directions

We currently have the concept of near (adjacent) neighbors and far neighbors (which exist with the implementation of the torus structure). This has implications for implementations on heterogeneous and distributed networks. Specifically, the far neighbors are metaphors for nodes on another processor in a distributed system. So, one component of the sort, partition, and send task could be that the data is partitioned not into equal subsets, but into subsets of a size proportional to the speed of the link to that node.

Furthermore, in heterogeneous architectures, the subset size could also be related to the speed of the corresponding neighbor node. Thus, future enhancements will include an applications kernel that will be resident on each node of the heterogeneous network. Upon startup, each kernel will negotiate with its near neighbor kernels to adjust the size of the exchange list (to be load balanced and sorted). The negotiated value will be a function of each node's own capacity in memory, processing, and its number of neighbors. Upon a fault, the kernels will re-negotiate the exchange files with the surviving near neighbors.

Acknowledgments

This research was partially supported by a grant from the Defense Nuclear Agency 93DNA-3. We gratefully acknowledge this support.

Reference

1. Gu, Qian Ping, and Jun Gu: Algorithms and Average Time Bounds of Sorting on a Mesh-Connected Computer. IEEE Transactions on Parallel and Distributed Systems. Vol 5, no 3. (March 1994) 308-315
2. Lin, Yen-Chun: On Balancing Sorting on a Linear Array. IEEE Transactions on Parallel and Distributed Systems. Vol 4, no 5. (May 1993) 566-571
3. Thompson, C.D., and H.T. Kung: Sorting on a Mesh-connected Parallel Computer. Communication of the ACM. Vol 20, no 40,. (April 1977) 263-271
4. Xu, Chengzhong and Francis Lau: Load Balancing in Parallel Computers: Theory and Practice. Kluwer Academic Publishers, Boston (1997)