High-Performace Sorting on Networks of Workstations

Meenali Rungta
CS 739: Distributed Systems
University of Wisconsin, Madison

1 March 2006
Spring 2006

1 Overview

The paper demonstrates that parallel sorting on a Network of Workstations (NOW) is competitive to sorting on large-scale SMPs.The clusters consist of commodity hardware and software, and the authors also evaluate the suitability of existing hardware, interfaces and mechanisms to the sorting application.

2 Problem Statement and Assumptions

The paper presents a NOW approach to the disk-to-disk sorting problem. Thus the authors show that NOWs are well-suited to I/O-intensive applications. *They have a single application in mind, and hence the system is relatively easier to build and optimize.* Also, the system is build on the following assumptions:

- The workstations in the cluster are connected to each other by high-speed switched ethernet.
- Though each workstation runs an independent unix based OS, the disparate resources are unified under a distributed OS like GLUnix. The authors use this primarily as a parallel program launcher.
- The communication layer in the nodes of the cluster is the lightweight ActiveMessage layer, built specially to take advantage of the low latency and high bandwidth of switch-based networks. *Normally, when a message arrives, TCP/IP buffers it till it reaches the application layer. The application then decides what has to be done with the arriving message. Active Messages, on the other hand, let the sender specify a handler with the message which is executed remotely and atomically with respect to other message arrivals. Since the various layers of TCP/IP are not present, the round-trip latency of the network is much less.*
- The authors measure the performance of NOW-Sort only on key values with uniform distributions. Further, they assume that the initial number of records on each workstation is equal.

3 Methodology

The authors develop a collection of sorting algorithms of increasing complexity, from a single-pass single node to the final two-pass parallel algorithm. They then implement the algorithm and study the system in detail to tune and configure it for the most optimal performance.

4 Design and Implementation

4.1 One-Pass Single-Node Sorting

The single-pass single node sort turns out to be the right 'building block' for the sorting application. In particular, this implementation contributes in studying the following:

- The optimal disk configuration: the authors develop a library to enable them to stripe larger blocks on

faster disks, to make use of the heterogenity in disk hardware. *This is expected to common phenomenon in large clusters, especially as once-deployed clusters have a mix of old and new disks in due course of time.*

The study suggests that out of the existing hardware, 4-disk configuration gives the best bandwidth. It is notable that later, in studying parallel version of sort, the authors find that due to communication and disk traffic over a common S-bus interconnect, it becomes a bottleneck and 2 disks over the wide SCSI turns out to be the best disk configuration. They however, use the 2-disk disks over fast-narrow SCSI due to hardware limitations. Such variablilty in optimal configurations of the variety of hardware in a cluster environment seems to be noteworthy.

- The usefulness of software (OS) interface: They find `mmap()` with `madvise()` adequate for reading records from disk, but need to develop a tool for determining the amount of available memory. The tool is useful to determine the number of records that can be sorted with a one-pass sort.

4.2 One-Pass Parallel Sorting

It's interesting to contrast the use of Active Messages as compared to Distributed File System in this case. NOW-Sort doesn't use a DFS at all while it uses distributed OS (GLUnix) mainly to launch parallel programs simultaneously. The use of ActiveMessage layer utilizes locality of data - the input files are read from local disks, keys are sent to remote buckets if needed using the communication layer, and final output files are written to local disks as well. A DFS, on the other hand, would read from remote files (unless information about locality is available), would send keys to remote buckets if needed, and write to local disks. A DFS built on Petal would probably not get to even write final files locally, as the virtual disk interface exposed by Petal completely abstracts the location information from a FS like Frangipani.

The authors also exploit overlapping of the two operations - reading from local disks and communicating with remote nodes. They develop synchronous, interleaved and threaded versions of the algorithm and observe that threaded (which overlaps reading and sending by means of separate threads) performs the best on a host of disk configurations. They also note that the previous optimal disk configuration of 4-disks doesn't go well at all with communication involved via the same S-bus due to latter's saturation, and that reading from 2-disks and writing to all 4 gives the new best configuration. *The saturation observed for S-bus is pretty acute: the theoretical peak is 80 MB/s, whereas the observed peak is 36 MB/s.*

4.3 Two-Pass Single-Node Sorting

This consists of two phases: A 'Create Runs' phase where the one-pass sort is repeated to create multiple sorted runs on disks, and a 'Merge' phase where sorted runs are merged into a single sorted file. Here the authors exploit the overlapping between read of a subsequent run with the write of the previous run in the 'Create Runs' phase. They also exploit the overlap between reading of records from all individual runs for merging and writing of part of the final sorted file.

4.4 Two-Pass Parallel Sorting

Here the authors explore overlapping reading and sending (as in one-pass parallel algorithm) and also reading and writing (as in two-pass single node sort). While all four operations being overlapped gives best performance in case of 2 disks, the S-bus is a point of contention in case of 4 disks (with read, write and network traffic all passing through it at the same time). Hence in 4-disks configuration, the best overlaps reading and sending, but performs reading and writing synchronously.

5 Evaluation

The paper evaluates the system incrementally, for every sorting implementation and for various configurations. It's interesting that NOWSort sets new record for MinuteSort (6 GB sorted in one minute on 64 processors) while doing twice the amount of I/O (sorting being done in 2 passes owing to insufficient memory) It also sets a new Datamation Benchmark record of 2.41 seconds on a 32-node cluster.

6 Conclusions

The paper successfully demonstrates the suitability of cluster environment for I/O intensive applications. The argument in favour of use of clusters for developing parallel applications is strong: they provide performance isolation (making it easy to determine the performance bottleneck) as compared to 'black-boxed' SMPs, and also enable incremental scalabilty (commodity hardware can be added easily and effectively to existing clusters).

However, NOWSort itself works only under idealized conditions - the issue of fault tolerance is not addressed at all. Since it utilizes soft state, re-sorting could be effectively used in case of node failures. However, issues like writing part of the same file twice could arise if writer thread fails in middle of it's operation. Also, as the cluster gets bigger, it becomes difficult to imagine that all machines would be up at the same time, and that all disks would be in perfect condition. Thus the techniques developed in the paper are a useful proof-of-concept but could not be readily applied to real world problems for lack of robustness.