An Evaluation of Architectural Alternatives for Rapidly Growing Datasets: Active Disks, Clusters, SMPs

Mustafa Uysal	Anurag Acharya	Joel Saltz
Dept. of Computer Science	Dept. of Computer Science	Dept. of Computer Science
University of Maryland	University of California	University of Maryland
College Park	Santa Barbara	College Park

Abstract

Growth and usage trends for several large datasets indicate that there is a need for architectures that scale the processing power as the dataset increases. In this paper, we evaluate three architectural alternatives for rapidly growing and frequently reprocessed datasets: active disks, clusters, and shared memory multiprocessors (SMPs). The focus of this evaluation is to identify potential bottlenecks in each of the alternative architectures and to determine the performance of these architectures for the applications of interest. We evaluate these architectural alternatives using a detailed simulator and a suite of nine applications. Our results indicate that for most of these applications Active Disk and cluster configurations were able to achieve significantly better performance than SMP configurations. Active Disk configurations were able to match (and in some cases improve upon) the performance of commodity cluster configurations.

1 Introduction

Growth and usage trends for several large datasets indicate that there is a need for architectures that scale the processing power as the dataset grows. The growth trends indicate that the rate at which several datasets are growing is outstripping the improvement in performance of commodity processors. The usage trends indicate that there is a change in user expectations regarding large datasets – from primarily archival storage to frequent reprocessing in their entirety.

Results from the 1997 and 1998 Winter Very Large Database surveys document the growth trends for decision support databases [40, 41]. For example, the Sears Roebuck and Co decision support database grew from 1.3 TB in 1997 to 4.6 TB in 1998. Patterson et al [29] quote an observation by Greg Papadopolous - while processors are doubling performance every 18 months, customers are doubling data storage every nine-to-twelve months and would like to "mine" this data overnight to shape their business practices [28]. Jim Gray argues that satellite data repositories will grow to petabyte size over the next few years and will require a variety of processing ranging from reprocessing the entire dataset to take advantage of new algorithms to re-projection and composition to suit different display requirements [17, 18]. Ferreira et al [14] estimate that digitizing a single slide under a high-resolution confocal light microscope requires between 35 and 200 GB. Plans for managing the pathology record of the Johns Hopkins Medical School call for digitizing tens of thousands of such slides. These images are to be used for telepathology, medical research and pedagogy and require a variety of processing including three-dimensional reconstruction of tissue sections, image segmentation, virtual staining and histological image analysis [3].

In this paper, we evaluate architectural alternatives for scaling the processing power with the growth in dataset size. We consider three alternatives: Active Disks [2, 19, 23, 31] (see section 2 for a brief review of Active Disks.), clusters and shared memory multiprocessors (SMPs). Each of these architectures can be incrementally scaled as the dataset size increases – Active Disks by adding disk units (and the embedded processors), clusters by adding new machines and SMPs, such as the SGI Origin-2000, by adding integrated modules with two-to-four processors. All three architectures are either currently in use for processing datasets of interest (relational databases, image databases, satellite data repositories) or have been proposed as suitable alternatives. Shared memory multiprocessors are widely used for relational databases (Strenstrom et al [35] estimate that in 2000, 40% of such machines will sold for handling relational databases). Clusters have been shown to provide excellent I/O performance: the current world-record for disk-to-disk sort (the Indy MinuteSort [20]) is held by NOW-sort running on a cluster [7]. Active Disks have been identified by several researchers as a cost-effective architectural alternative for applications that process rapidly growing datasets [2, 19, 23, 31].

To evaluate these architectures, we use a suite of nine applications that process datasets of interest: (1) SQL select, (2) SQL aggregate, (3) SQL `group-by`, (4) external sort, (5) the *datacube* operation for decision support [21], (6) SQL join, (7) datamining retail data for association rules [5], (8) image convolution, and (9) generation of earth images from raw satellite data [12, 13, 34]. The first seven applications process relational databases and are used in data warehouses; the remaining two are used in image databases and satellite data repositories respectively. These applications vary in characteristics such as the amount of computation per byte of I/O, the number of times the entire dataset is read, whether intermediate and final results are written to disk and whether a disk-to-disk shuffle of the input dataset (or a part thereof) is performed. We believe that, taken as a group, these applications are representative of the applications that process the datasets of interest.

Based on our experiments, we tried to answer two questions. First, for each of these architectures, does the performance of the test applications scale with configuration size? We used the number of disks (and processors) as a measure of the configuration size. If not, which component of the architecture (processors, disks, I/O interconnect, network interconnect) becomes a bottleneck? Second, how does the performance of all three architectures compare for the applications? For comparison between architectures, we configured each of them with identical disks and used configurations with equal number of disks (and processors).

Our results indicate that for most of these applications Active Disk and cluster configurations were able to achieve significantly better performance than SMP configurations. Active Disk configurations were able to match (and in some cases improve upon) the performance of commodity cluster configurations. To be able to effectively handle applications that redistribute their input dataset, such as external sort and distributed join queries, Active Disks require the ability to communicate directly with peers. Requiring all communication to pass through the front-end host can lead to substantial loss of performance for these applications. Given the substantial impact for important applications, we revise our original proposal for Active Disk architectures [2] to include direct disk-to-disk communication.

2 Background: Active Disks

In this section, we provide a brief introduction to Active Disks. Active disks integrate significant processing power and memory into a disk drive and allow application-specific code to be downloaded and executed on the data that is being read from (written to) disk. To utilize Active Disks, an application is partitioned between a host-resident component and a disk-resident component. The key idea is to offload bulk of the processing to the disk-resident processors and to use the host processor primarily for coordination, scheduling and combination of results from individual disks.

Acharya et al [2] propose a stream-based programming model for the disk-resident component (*disklet*) and its interaction with host-resident peer. Disklets take streams as inputs and generate streams as outputs. Files (and ranges in files) are represented as streams. Streams are accessed using a standard interface which delivers the data in buffers whose size is known apriori. A disklet can be written in any language. However, it is required to adhere to certain guidelines. A disklet cannot allocate (or free) memory. It is sandboxed [39] within the buffers corresponding to each of its input streams, which are allocated and freed by the operating system, and a scratch space that is allocated on its behalf when it is initialized. A disklet is also not allowed to initiate I/O operations on its own. These restrictions limit the amount of damage that can be done by a disklet. They also simplify the operating system support required on disk-processors and help reduce its memory footprint.

Active Disks require a thin layer of operating system support (the *DiskOS*) at the disk. The DiskOS provides three services – memory management, stream communication and disklet scheduling. The stream-based model simplifies memory management as all memory is allocated in contiguous blocks whose size is known a priori and the lifetime of all blocks is known. The stream-based model also simplifies the communication support required as all stream buffers are allocated and managed by the DiskOS. Depending on the amount of memory available, it can allocate multiple buffers and overlap data movement and computation. The stream-based model also simplifies scheduling for disklets. A disklet is ready to run whenever there is new data available on one or more of its input streams.

3 Methodology

To conduct these experiments, we developed a simulator called *Howsim* which simulates all three architectures. *Howsim* contains detailed models for disks, networks and the associated libraries and device drivers and relatively coarse-grain models of processors and I/O interconnects.

For modeling the behavior of disk drives, controllers and device drivers, *Howsim* uses the *Disksim* simulator developed by Ganger et al [15]. *Disksim* has a detailed disk model that supports zoned disks, spare regions, segmented caches, defect management, prefetch algorithms, bus delays and control overheads. *Disksim* has been validated against several disk drives using the published disk specifications and SCSI logic analyzers; it achieves high accuracy - the worst case *demerit figure* [32] for *Disksim* is only 2.0% of the corresponding average response time [15]. For modeling I/O interconnects, *Howsim* uses a simple queue-based model that has parameters for startup latency, transfer speed and the capacity of the interconnect.

For modeling the behavior of networks, message-passing libraries and global synchronization operations, *Howsim* uses the *Netsim* customizable network simulator developed by Uysal et al [38]. *Netsim* models switched networks and an efficient user-space message-passing and global synchronization library with an MPI-like interface. *Netsim* has been validated using a set of microbenchmarks on an IBM SP-2 with a multistage proprietary switch and a 10-node Alpha SMP cluster with an ATM switch yielding 2-6% accuracy for most messages.

For modeling the behavior of user processes, *Howsim* uses a trace of processing times and I/O requests for individual tasks. It models variation in processor speed by scaling these processing times. To acquire the traces of processing time for user-level tasks, we implemented each application on a DEC Alpha 2100 4/275 workstation with 256 MB of memory. We ran each application with the same dataset and I/O request sizes as used in our experiments. For applications that use the amount of memory available as an explicit parameter (Sort, Join and Datacube), we generated traces for multiple memory sizes – to allow us to simulate architectures with different amounts of memory.

For modeling operating system behavior on hosts, *Howsim* uses parameters that represent the time taken for individual operations of interest: `read/write` system calls, context switch time, the time to queue an I/O request in the device-driver and the time to service an I/O interrupt. We obtained the first two using `lmbench` [25] on a 300MHz Pentium II running Linux (10μs for read/write calls, 103μs for context-switch). We charged a fixed cost of 16μs to queue an I/O request in the device-driver.

For Active Disks, *Howsim* models a preliminary implementation of *DiskOS* which provides support for scheduling disklets as well as for managing memory, I/O and stream communication. It uses a modified version of *DiskSim* that is driven by the disk operating system layer. Disklets are written in C and interact with *Howsim* using a stream-based API [2]. `Howsim` has additional parameters for the DiskOS. For this study, we assumed the system call and context switch costs on the DiskOS to be 1 μs. In addition, another 1 μs is charged to initiate a disk request from DiskOS and to service an interrupt from the disk mechanism. Given that disklets execute within the same protection domain as the DiskOS, we believe that these costs are reasonable.

For clusters, *Howsim* uses *Netsim* to model the interconnect and *DiskSim* to model the I/O subsystem. An MPI-like message passing interface is used to drive the network model, providing point-to-point communication and global reduction operations. The disk model is driven by a raw-disk access library.

For SMPs, *Howsim* models two-processor boards connected by a low latency, high bandwidth interconnect. For communication, it models one-way block-transfers, `shmemget/shmemput`, as available on the Origin machines as well as the Cray T3D/T3E. Block transfers are suitable for the applications under consideration as they move large volumes of data in relatively large chunks. For synchronization on SMPs, *Howsim* provides spin-locks, remote queues [10] and global barriers. We used at-memory fetch-and-op primitive as provided by SGI Origin for spin-locks (which cost around 3μs [22]). *Howsim* models a high-bandwidth I/O subsystem similar to the XIO subsystem available in the Origin 2000. The disk model is driven by a striping library written on top of the raw-disk access library.

4 Architectures and configurations

We had two goals for our experiments: first, to evaluate the performance of each architecture as the processing power and dataset size is scaled; and second, to compare the performance of comparable configurations of each architecture. We scaled the configurations for each architecture in disk-processor pairs: Active Disk configurations were scaled by adding disk units (each with an embedded processor); cluster configurations were scaled by adding new hosts, each with a single processor and a single disk; SMP configurations were scaled by jointly adding four-processor modules (as in the SGI Origin 2000) and four-disk sets.

For comparison between architectures, we configured each of them with identical disks and we used configurations with equal number of disks (and processors). For the rest of the components, we follow the configuration guidelines suggested by experts. For each architecture, we defined configurations with 16, 32, 64 and 128 disks (and processors). To understand the impact of scaling individual components and to identify the bottleneck resources for individual applications, we performed additional experiments by selectively scaling individual components.

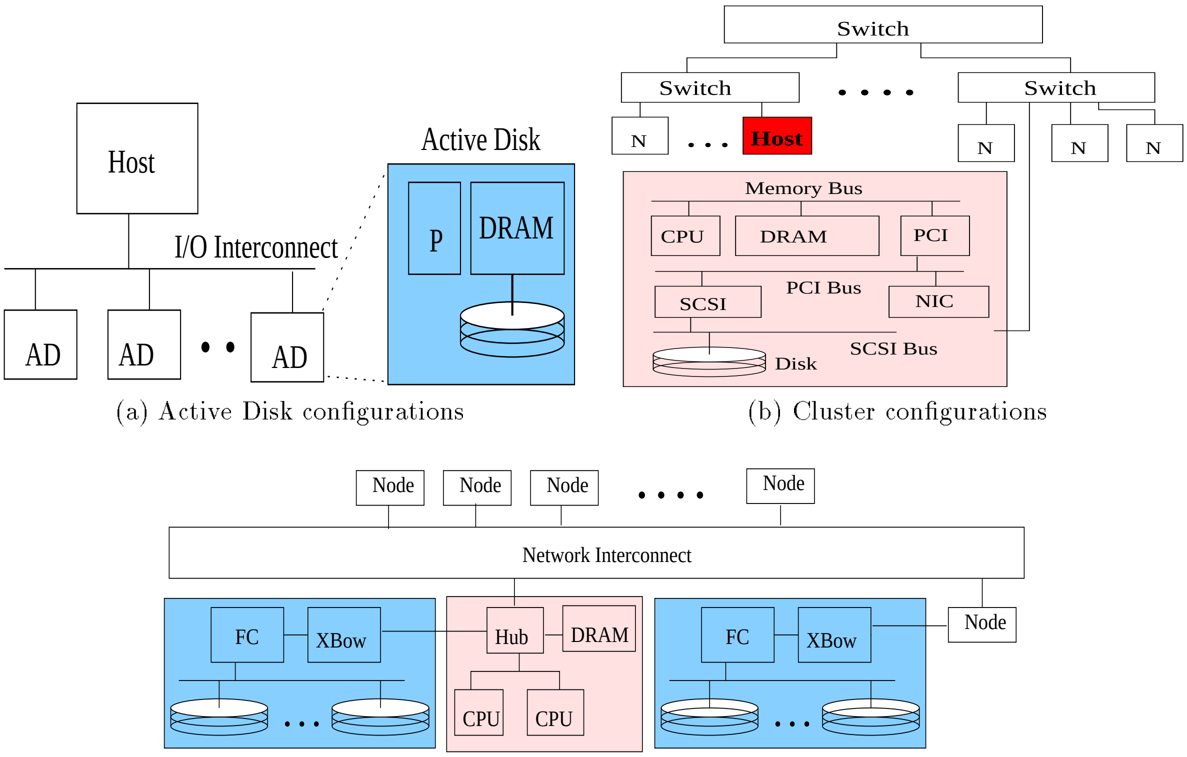
For all configurations, we assumed disks similar to the Seagate 39102FC from the Cheetah 9LP disk family [33]. These disks have a spindle speed of 10,025 rpm, a formatted media transfer rate of 14.5-21.3 MB/s, an average seek time of 5.4 ms/6.2 ms (read/write) and a maximum seek time of 12.2 ms/13.2 ms (read/write). They support Ultra2 SCSI and dual-loop Fiber Channel interfaces.

Active Disks: For the Active Disk configurations, we assumed that: (1) a Cyrix 6x86 200MX processor (200 MHz) and 32 MB of 10ns SDRAM were integrated in the disk units; (2) all the disks were connected by a dual-loop Fiber-channel interface with a bandwidth of 200 MB/s (100 MB/s per loop); (3) the disks can directly address and communicate with each other using a SCSI-like interface; and (4) communications with clients are handled by a front-end host with a 450 MHz Pentium II and 1 GB RAM. Figure 1 (a) illustrates Active Disk configurations.

To identify the bottleneck resources for individual applications, we studied alternative configurations that individually scaled: (1) the aggregate bandwidth of the serial interconnect to 400 MB/s; (2) the speed of the processor in the front-end host to 1 GHz; and (3) the memory integrated into the disk unit to 64 MB and 128 MB. No software changes were associated with scaling the bandwidth of the serial interconnect and speed of the front-end processors. To take advantage of the additional memory available in the 64 MB/disk and 128 MB/disk configurations, the number of buffers allocated per stream by the DiskOS was increased from two to four and eight respectively. This allowed these configurations to tolerate longer communication and I/O latencies. Finally, to understand the impact of allowing the disks to communicate directly with each other, we considered alternative configurations that restrict disks to communicate only with the front-end host (as proposed in [2, 31]).

Clusters: For the cluster configurations, we assumed that each host contained: (1) a 300 MHz Pentium II, (2) 128 MB of 10ns SDRAM, (3) a 133 MB/s PCI bus, and (4) a 100BaseT ethernet NIC. We further assumed that the hosts were connected to 24-port 100BaseT ethernet switches with two gigabit ethernet uplinks similar to the 3Com SuperStack II 3900 [27, 36] – the 16 host configuration being connected to a single switch, larger configurations being connected to an array of switches with the uplinks connecting to a gigabit ethernet switch similar to the 3Com SuperStack II 9300 [36, 37]. Figure 1 (b) illustrates cluster configurations. We selected these configurations based on the design of large Beowulf-class clusters with commodity components (e.g. Avalon [9]).

We assumed that these machines ran a standard full-function operating system similar to Solaris. Acharya et al [1] report that, on the average, the kernel on a 128 MB Solaris machine has a memory footprint of 24 MB (including the paging free list but not including the file cache). Accordingly, we assumed that only 104 MB on these hosts is available to user processes. We assumed these machines provided an efficient user-space messaging and synchronization library similar to BSPlib [11] that pins send/receive buffers on every host for every communicating peer. We also assumed an asynchronous I/O interface like `lio_listio`. To



(a) Active Disk configurations

(b) Cluster configurations

(c) SMP configurations

Figure 1: Schematics of the three architectures.

determine the impact of varying the network bandwidth, we studied alternative configurations that scaled the bandwidth of the network interconnect to 1 Gbps per host.

Shared memory multiprocessors (SMPs): For the SMP configurations, we followed the guidelines for configuring decision support servers (as quoted by [23]): (1) put as many processors in a box as possible to amortize the cost of enclosures and interconnects; (2) put as much memory as possible into the box to avoid going to disk as much as possible; and (3) attach as many disks as needed for capacity and stripe data over multiple disks to quickly load information into memory. We assumed an SMP configuration similar to the SGI Origin 2000: (1) two-processor boards (with 250 MHz processors) that directly share 128 MB memory; (2) a low-latency high-bandwidth interconnect between these boards ($1\mu s$ latency and 780 MB/s bandwidth); (3) a high-performance block-transfer engine (521 MB/s sustained bandwidth [24]); (4) a high-bandwidth I/O subsystem (two I/O nodes with a total of 1.4 GB/s bandwidth), similar to XIO, that connects to the network interconnect; and (5) a dual-loop Fiber Channel I/O interconnect (200 MB/s) for all disks. Figure 1 (c) illustrates the SMP configurations. Note that the amount of memory is scaled with the number of processors – a 64-processor configuration having 4 GB and a 128-processor configuration having 8 GB.

We assumed that these machines ran a standard full-function operating system like IRIX and provided the `lio_listio` asynchronous I/O interface and user-controllable disk striping for individual files. Further, we assumed that these machines provided a *remote queue* abstraction (as suggested by Brewer et al [10]). To identify the bottleneck resources for individual applications, we studied alternative configurations that individually scaled the bandwidth of the serial I/O interconnect to 400 MB/s.

5 Applications

Our suite of applications consists of nine applications from three application domains - relational databases, image databases and satellite data repositories. For each application, we started with a well-known efficient algorithm from the literature and adapted it for each architecture and the corresponding programming model.

For Active Disks, we adapted the algorithms to use the stream-based programming model proposed by Acharya et al [2]. Note that, overlapping computation and communication is handled by the DiskOS (the disk-resident OS layer) by using multiple buffers per stream.

For clusters, we adapted the algorithms to use MPI-like asynchronous message-passing operations and global synchronization primitives. Each host posts up to 16 asynchronous receives for *any* message from *any* peer. We adapted all algorithms to use large (256 KB) I/O requests and deep request queues (up to four asynchronous requests) to take full advantage of the aggressive I/O subsystem and to overlap the computation with the I/O as much as possible. Since each host can only address its own disk, we partitioned the input datasets over all hosts.

For SMPs, we adapted the algorithms to use one-way block-transfers (`shmemput/shmemget`) and remote queues for moving data between processors. Given the volume of data being transferred and the one-way nature of the data movement, block-transfers and remote queues are suitable for these applications. We striped each file over all disks using a 64 KB chunk per disk. To take advantage of the aggressive I/O subsystem, each processor issues up to four 256 KB asynchronous requests (each request transferring 64 KB from four disks). Note that for `sort` and `join`, which shuffle their entire dataset and write it back to disk, we partitioned the disks into separate read and write groups (as in NOW-sort [7]). Since all processors can address all disks, we did not a-priori partition the input datasets to processors. Instead, we maintained two shared queues (read/write) of fixed-size blocks in the order they appear on disk. When idle, each processor locks the queue and grabs the next block off the queue. This technique reduces the seek costs at the disks as the overall sequence of requests roughly follows the order in which data has been laid out on disk. A-priori partitioning of the dataset would result in a potentially long seek for every request.

SQL `select` and `aggregate`: these are simple one-pass algorithms – `select` filters tuples from a relation based on a user-specified predicate and `aggregate` computes a single aggregate value for all tuples in a relation.[1] The active-disk algorithm performs the filtering/aggregation locally and forwards the results to the front-end host. The front-end concatenates/aggregates data from different disks. The cluster algorithm and SMP algorithm are similar. In the former, each host performs filtering/aggregation on its partition of the data and forwards the results to the front-end; in the latter, each processor dynamically selects 256 KB chunks from the input relation and directly writes the results to the destination buffer using block-transfer. Both `select` and `aggregate` perform little computation/byte.

SQL `group-by`: The `group-by` operation computes a one-dimensional vector of aggregates indexed by a list of attributes [26]. It partitions a relation into disjoint sets of tuples based on the value(s) of index attribute(s) and computes an aggregate value for each set of tuples. We used the hashing-based algorithm from [16] as the starting point. The active-disk and cluster algorithms are similar. They perform the `group-by` in two steps. In the first step, each disk/host performs local `group-by`s as long as the number of aggregates being computed fits in its memory. When it runs out of space at a disk/host, it ships the partial results to the front-end and reinitializes its memory. The front-end accumulates the partial results. In the SMP algorithm, each processor computes a local version of the `group-by`; results from all processors are merged at the end. Note that, in our experiments, `group-by` generates significantly larger results than `select`. It also performs more computation/byte as it needs to maintain a hash-table of aggregates.

Datacube: the `datacube` is the most general form of aggregation for relational databases. It computes multi-dimensional aggregates that are indexed by values of multiple aggregates [21]. In effect, a datacube computes group-bys for all possible combinations of a list of attributes. We used the *PipeHash* algorithm proposed in [4] as the starting point for our algorithms. It schedules the group-bys as a sequence of pipelines; all the group-bys in a pipeline are computed as a part of a single scan of disk-resident data. The final results of each pipeline are stored back on disk; some of these results are used as input for following pipelines. For

[1] Using one of the five SQL aggregation operations: `min`, `max`, `sum`, `avg` and `count`.

individual group-bys, PipeHash uses a hashing-based technique [16]. The active-disk and cluster algorithms are similar. For every pipeline, they partition the memory available at each disk/host in proportion to the estimated size of the group-bys being performed in the pipeline. For each group-by, they partition the range of values over all the disks/hosts; each disk/host is responsible for combining results from all peers for that range of values. Each disk/host performs local `group-bys` as long as the number of aggregates being computed fits in its memory. When it runs out of space, it partitions the partial results and ships each partition to the disk/host that is responsible for the corresponding range of values. The SMP algorithm performs the group-bys in a batched manner – similar to that for `group-by`. After all the results for a group-by have been accumulated, the result is written to disk. Note that since `datacube` performs multiple group-bys in a single scan, it performs more computation per byte read than `group-by`. Also, since it computes a multi-dimensional aggregate, it generates and communicates significantly more data.

External sort: we used the two-pass parallel `NOW-sort` [7] as the starting point for our sort algorithms. The active-disk and cluster algorithms are fully pipelined in that they overlap reading data, sending data to peers and sorting and writing data. The SMP algorithm overlaps just the first two operations; reading and writing operations are performed synchronously (Dusseau et al [7] recommend that for less than four disks, all operations should be overlapped whereas for more than four disks, only the first two should be overlapped). The first pass of these algorithms repartitions their entire input on disks and has large communication requirements. The second pass for the active-disk and cluster algorithms is localized; each disk/host operates on its own partition. The SMP algorithm uses a disjoint set of disks for reading and writing in the first phase – it divides the total number of disks into two for this purpose. Note that the first pass of `sort` is communication-intensive and requires all-to-all communication. Since it repartitions its entire dataset, `sort` performs significantly more communication than `datacube`.

Project-Join query: we used a sort-merge join for this application. A sort-merge join partially sorts each of the relations being joined and performs a join by stepping through the partially sorted relations using a pair of loops. We based our join algorithms on the two-pass NOW-sort. The first two passes of these algorithms are similar, in structure, to the first pass of a two-pass sort: the first pass repartitions and creates sorted *runs* for the first relation; the second pass does the same for the second relation. The third pass of these algorithms is similar to the second pass of a two-pass sort: it maintains a heap for the heads of the sorted runs for each relations and performs the join by picking elements from the two heaps. The first two passes of these algorithms have large communication and I/O requirements. The third pass for active-disk and cluster algorithms is localized as each disk/host operates on its own partition. The SMP algorithm uses a disjoint set of disks for reading and writing in the first two passes – it divides the total number of disks into two for this purpose. Note that the first pass of `join` is communication-intensive and requires all-to-all communication. Since both `sort` and `join` repartition their entire dataset, their communication requirements are similar.

Datamining: we focus on frequent itemset counting for mining association rules in retail transaction data [5]. We used the `eclat` algorithm [42] as the starting point for our algorithms. It is a multi-pass algorithm with the first two passes same as the *Count distribution* algorithm proposed by Agrawal et al [6]. After the first two passes, it clusters the candidate itemsets into equivalence classes and uses these classes to filter, transpose and repartition the input data sets. The third pass is localized and does not require any communication. It is also I/O-optimized as each processor is able to perform all its remaining computation with a single scan of its partition. Unlike external sort and sort-merge join, this application repartitions only a fraction of its input dataset (the exact fraction depends on the parameters the algorithm is run with). The *eclat* algorithm was originally described for shared memory multiprocessors. We adapted it for Active Disks and clusters by reverting to *Count distribution* in the first two passes. The original SMP algorithm performed fine-grained updates; we modified it to batch updates to the counters associated with itemsets. The original algorithm built a large triangular array of counters in its second pass. We noticed that a large fraction of the elements were zero in all our experiments and optimized it for memory consumption by using a sparse array. Note that `dmine` needs to communicate only the counters in the first two passes and a significantly reduced version of its input data in the third pass. Its communication requirements, therefore, are smaller than that of `sort`, `join` and `datacube`.

Image convolution: convolution is widely used to enhance spatial features or subdue noise in images. It computes a new value of each pixel as a linear combination of its own value and the values of its neighboring pixels. The coefficients for the linear combination are specified as a matrix (known as the *kernel*). The active-disk and cluster algorithms partition the images over all disks; the SMP algorithm stripes a file with all images over all disks, each stripe containing an integral number of images. The active-disk and cluster algorithms process each image independently and forward it to the front-end; each processor in the SMP algorithm stores the processed image directly using block-transfers. Note that `conv` is compute-intensive and performs a very large amount of computation per byte read.

Generating composite satellite images: Earth scientists generate earth images by compositing remotely-sensed data acquired over multiple days from satellite-based sensors. Generating a composite image requires pre-processing and projection of the sensor values onto a two-dimensional grid followed by composition of all values that map onto a single grid point to generate the associated pixel. We base our algorithms on the technique used in several programs used by NASA [12, 13, 34]. All our algorithms process sensor values in large chunks, mapping each value to the output grid and performing the composition operation using an accumulator for every output pixel. The active-disk and cluster algorithms perform local accumulation to reduce the amount of data communicated. The output image for the high-resolution datasets (about 556 MB [34]), however, does not fit into the memory available at individual disks/hosts. To deal with this, each disk/host performs accumulation for a contiguous section of the output grid that fits into its memory. When a sensor value that maps outside this subgrid is encountered, the partial result is shipped to the front-end. As each partial result is received at the front-end, it is composed into the final image. The SMP algorithm uses a similar technique – each processor performs local accumulation as long as possible; when local accumulation is no longer possible, it locks and `shmemget`s the corresponding portion of the global grid, merges the data from the local subgrid and `shmemput`s the final result back to the final location. Note that `earth` is compute-intensive and performs a very large amount of computation per byte read.

6 Datasets

We used 16 GB datasets for all the applications except `join` for which we used a 32 GB dataset. In this section, we describe the structure of the datasets for the different applications.

Select, Aggregate, Group-by: for these applications, we used a dataset with about 268 million tuples, each tuple being 64 bytes. For `select`, we test a single 4-byte field with a selectivity of 1%. For `aggr` and `group-by`, we used the `sum` operation on a 4-byte field. For `group-by`, we used a 4-byte field with 13.5 million distinct values as the grouping attribute.

Datacube: for `dcube`, we used a dataset with 536 million tuples. Each tuple had eight 4-byte attributes. We used four attributes as group-by attributes and the remaining four as aggregation attributes with `sum` as the aggregation function. The number of distinct values for each of the group-by attributes were 5.36 million, 536,000, 53,600 and 5,360. We created this dataset by scaling one of the datasets used in the paper that described the *PipeHash* algorithm [4].

Sort: for `sort`, we used a dataset with 100-byte tuples and 10-byte uniformly distributed keys. The total number of tuples was about 170 million. We created this dataset based on the standard sort benchmark described in [20].

Project-Join: for `join`, we used a dataset with 64-byte tuples and 4-byte uniformly distributed keys. The projection operation extracted eight 4-byte fields from each tuple. Each relation was 16 GB and contained 268 million tuples. The output for `join` was about 108 MB.

Datamining: for `dmine`, we used a dataset with 300 million transactions. The total number of items was 1 million and the average length of the transactions was 4 items. We generated this dataset using the Quest datamining dataset generator which we obtained from IBM Almaden [30]. For generating the frequent itemsets, we used a *minimum support* parameter of 0.001 (0.1%).

Image convolution: for `conv`, we used a dataset consisting of 65536 512x512 grayscale images with one byte per pixel. We used a 5x5 convolution kernel.

Earth science: for `earth`, we used a dataset which corresponds to high-resolution AVHRR images from the NOAA polar-orbiting satellites [34]. The output image for this dataset was 556 MB.

7 Results

Figure 2 compares the performance of all nine applications on comparable configurations of all three architectures. The results for each application on configurations of a particular size (16/32/64/128) are normalized with respect to the performance of the same application on the Active Disk configuration of the same size. We make six observations:

- The performance of `select` and `aggregate` is dominated by time taken to move the data from the disks to the processors. For all sizes, SMP configurations take longer to move data to the processors as their I/O interconnect (two Fiber-Channel loops) is unable to keep up with the I/O demand. The performance of Active Disk and cluster configurations is the same for all sizes except the 128-node configurations. In this case, `select` is slower on clusters due to serialization at the front-end – with its 100BaseT link, the cluster front-end can receive at no more than 10 MB/s whereas the Active Disk front-end with its Fiber-Channel link can receive up to 200 MB/s.

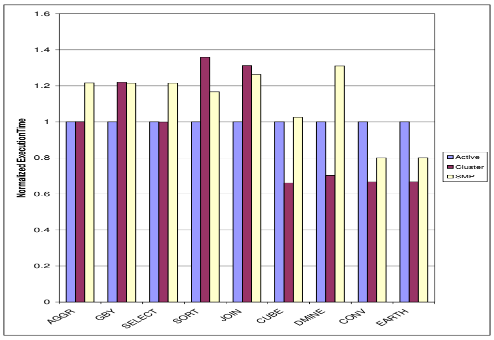
- The performance of `group-by` is significantly better on Active Disk configurations than on cluster or SMP configurations, albeit for different reasons. On SMP configurations, the performance of `group-by` is limited by bandwidth of the I/O interconnect; on cluster configurations, its performance is limited by serialization at the front-end.

- The performance of `sort` and `join` is significantly better on Active Disk and cluster configurations than on SMP configurations. On SMP configurations, both `sort` and `join` move their data four times over the I/O interconnect. The Fiber Channel loops used by the SMP configurations are not able to keep up with the I/O demand. In comparing the performance of these applications on Active Disk and cluster configurations, we note a pattern. For `sort`, the Active Disks perform better for 16/32 node configurations, the two are approximately equal for 64 node configurations and clusters perform better for 128 node configurations. This is due to the all-to-all communication performed by `sort` to repartition its input data. For configurations smaller than 32, total all-to-all communication bandwidth for clusters is $\leq$ 160 MB/s (8 or 16 pairs); whereas the the corresponding bandwidth for Active Disks is 200 MB/s. In contrast, for a 128-node cluster, total all-to-all communication bandwidth is 640 MB/s (64 pairs).

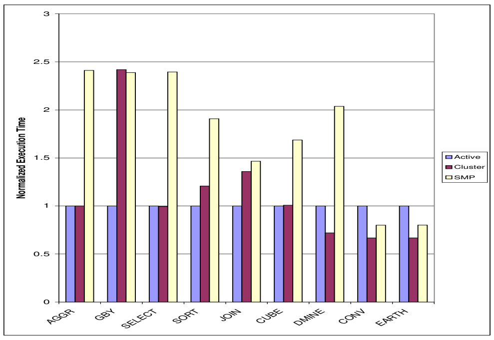
- The performance of `datacube` is significantly better on Active Disk and cluster configurations than on SMP configurations. On SMP configurations, `datacube` moves its input data once and each group-by twice over the I/O interconnect. The performance of `datacube` is roughly equal on Active Disks and clusters for 32 and 64 nodes. On 16 nodes and 128 nodes, cluster configurations achieve better performance, albeit for different reasons. For 16 nodes, the Active Disk configuration does not have enough memory to execute even a single pipeline within the disks. Therefore, it has to frequently forward the results to the front-end. For 128 nodes, the cluster configurations achieve better performance due to their greater communication capability.

- The performance of `dmine` is significantly better on Active Disk and cluster configurations than on SMP configurations. On SMP configurations, `dmine` moves its entire dataset twice and a filtered version of the dataset once over the Fiber Channel which is not able to keep up with the I/O demand. On Active Disk and cluster configurations, the performance of `dmine` is dominated by computation time (since the first two passes communicate just itemset frequency counters and the third pass communicates a filtered version of the input). Accordingly, the ratio of its performance on these architectures is determined by the ratio of their processor speeds.

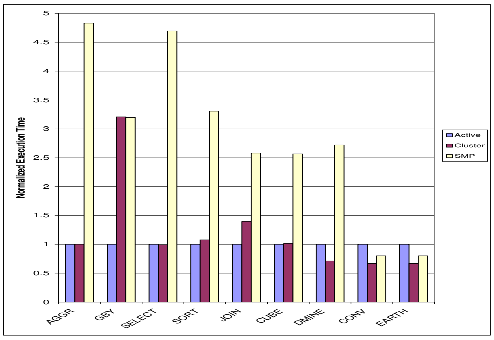
- The performance of `conv` and `earth` is dominated by the computation time for all architectures and configurations. For each configuration size, the relationships between their performance on the three architectures is determined solely by the relationships between the speed of the corresponding processors (200MHz for Active Disks, 250MHz for SMP and 300MHz for clusters).



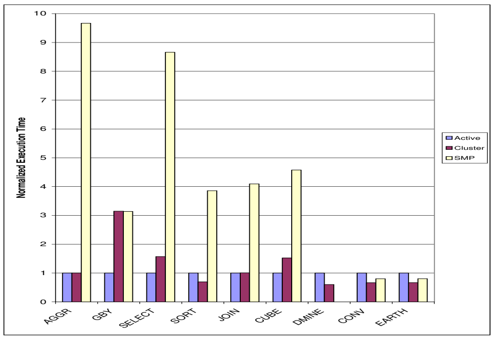
(a) 16 disk configurations



(b) 32 disk configurations



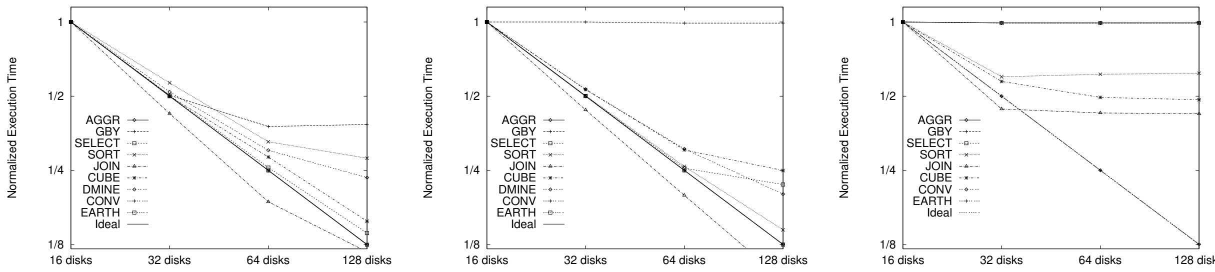
(c) 64 disk configurations



(d) 128 disk configurations

Figure 2: Performance of all nine applications on comparable configurations of all three architectures. The results for each application on configurations of a particular size (16/32/64/128) are normalized with respect to the performance of the same application on the Active Disk configuration of the same size. Note that the range of the y-axis is *different* for every graph. *The bar for* `dmine` *on the 128-node SMP configuration is missing as we could not complete the simulation in time.*

Figure 3 shows how the performance of each application scales for each architecture. The performance of most applications saturates relatively early on SMP configurations (other than the two computation-intensive applications, `conv` and `earth` which scale perfectly) . Applications whose performance is dominated by I/O, such as `select`, `aggregate` and `group-by` achieve no benefit from larger configurations. Application performance scales better on Active Disks and clusters: (1) the performance of `sort` saturates at 64 nodes for both architectures; (2) the performance of `dmine` saturates earlier for Active Disks (I/O interconnect becomes a bottleneck); whereas (3) the performance of `group-by` saturates earlier for clusters (the network link to the front-end becomes a bottleneck). Note that `join` seems to achieve better than perfect scaling for all architectures. We suspect that this is due to cache capacity effects during the merge phase. A similar result was reported by Arpaci-Dusseau et al [7] for NOW-sort with more than 17 runs per node (our experiments had between 2 and 80 runs per node).



(a) Active Disks (b) Clusters (c) SMPs

Figure 3: Variation in performance of all nine applications with configuration size for all three architectures. The results for each application on a given architecture are normalized with respect to the performance of the same application on the 16-node configuration of the same architecture. The `ideal` curve is for reference and indicates perfect scaling. Note that `join` seems to achieve better than perfect scaling for all architectures. We suspect that this is due to cache capacity effects during the merge phase.

Figure 4 examines the impact of varying individual components in Active Disk configurations. Figure 4 (a) presents the impact of restricting disks to communicate only with the front-end. We note that this restriction has a large impact (up to a five-fold slowdown) for the three communication-intensive applications and virtually no impact on the remaining six applications. Given the substantial impact for important applications, we revise our proposal for Active Disk architectures to include direct disk-to-disk communication.

Figure 4 (b) presents the impact of doubling the bandwidth of the I/O interconnect to 400 MB/s. We note that only the three communication-intensive applications derive benefit from it; the other six are unable to take advantage of the increased bandwidth. For the communication-intensive applications, `sort` on 128 nodes achieves close to a linear speedup, the other two applications achieve significantly smaller gains.

Figure 4 (c) presents the impact of upgrading the front-end host to a 1 GHz processor. We note that this mainly benefits applications that deliver substantial amounts of data to the front-end (for delivery to the requesting client). The speedup is close to linear for `group-by` which delivers significantly more data than `select`, for which the speedup is small. In addition to these applications, `dmine` achieves a speedup of up to 1.4 (for 128 disks) since it uses the front-end host to combine the itemset frequency counters forwarded by each disk.

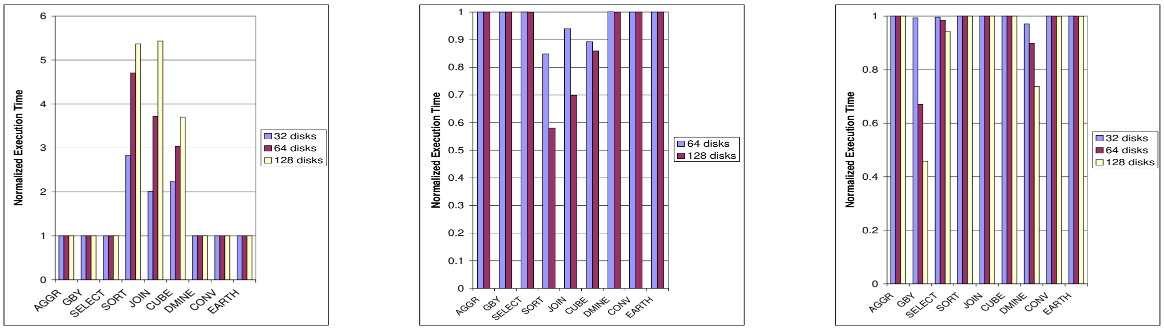
Figure 5 examines the impact of varying the bandwidths of bottleneck interconnects in cluster and SMP configurations. Figure 5 (a) presents the impact of varying the network bandwidth in cluster configurations. We note that most applications can derive significant benefit from an network interconnect with more bandwidth. Applications that saturate the link to the front-end host, `group-by` and `select`, derive the largest benefit. Applications that perform all-to-all communication (`sort`, `join`, `datacube` and `dmine`) also achieve significant benefit.

Figure 5 (b) presents the impact of doubling the bandwidth of the I/O interconnect in SMP configurations to 400 MB/s. We note that all applications except `conv` and `earth` achieve significant speedups. For 128-node configurations, all these applications, except `join` which requires a lot of seeks, achieve linear or close to linear speedups.

8 Discussion and conclusions

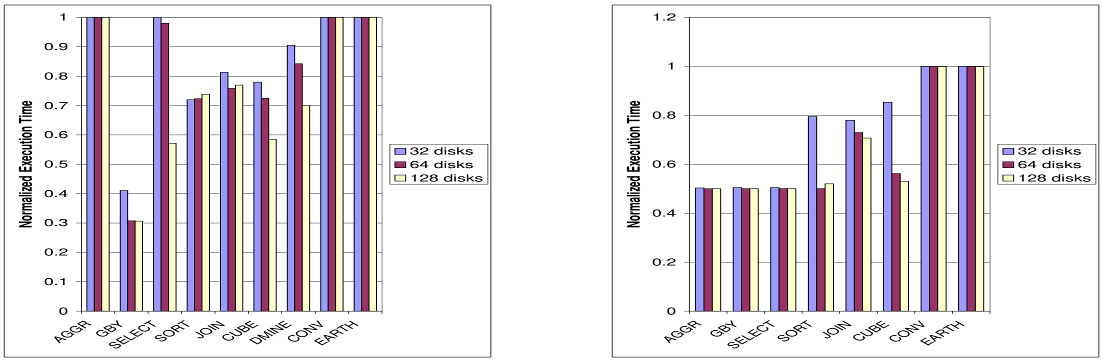
In this section, we present and discuss the conclusions of our study. We also compare the estimated prices of the configurations under study.

There are four main conclusions of our study. First, for most applications that process rapidly growing datasets, the Active Disk and cluster configurations were able to achieve significantly better performance than the SMP configurations. The performance of the SMP configurations used in our experiments was



(a) Restricted communication (b) 400MB/s interconnect (c) 1 GHz front-end

Figure 4: Impact of varying individual components in Active Disk configurations. The results for each application on a given configuration are normalized with respect to the performance of the same application on the core configuration of the same size.



(a) 1 Gbps network in clusters (b) 400 MB/s I/O interconnect in SMPs

Figure 5: Impact of varying the network interconnect in clusters and the I/O interconnect in SMPs. The results for each application on a given configuration are normalized with respect to the performance of the same application on the corresponding core configuration of the same size.

limited by I/O interconnect bandwidth, particularly for large configurations. The I/O interconnect sits in between all the processors and all the disks; the data for many applications (in particular decision support applications) passes over it multiple times. While it is possible to increase the I/O interconnect bandwidth for SMP configurations beyond what we assumed, it nevertheless remains a potential bottleneck as the number of disks increases.

Second, Active Disk configurations were able to match (and in some cases improve upon) the performance of commodity cluster configurations for I/O-bound and communication-bound applications. The high-bandwidth Fiber Channel interconnect used in Active Disks is the key reason why Active Disk configurations with slower embedded processors are able to, at least in some cases, outperform cluster configurations with faster processors with Fast Ethernet links.

Third, to be able to effectively handle applications that redistribute their input dataset, such as external sort and distributed join queries, Active Disks require the ability to communicate directly with peers. Requiring all communication to pass through the front-end host can lead to substantial loss of performance for these applications (a three-fold slowdown on 32-disk configurations, a five-fold slowdown on 128-disk configurations). Given the substantial impact for important applications, we revise our original proposal for Active Disk architectures [2] to include direct disk-to-disk communication. We also extend the stream-based programming model proposed in [2]to allow the host to establish connections between disklets running on different disks. Note that the sources and sinks for all streams are still specified by the host and disklets are still not allowed to determine (or change) where its input streams come from or where its output streams go to.

Four, we note that adding more memory to Active Disks provides limited advantage. Our results indicate that increasing the disk-memory to 128 MB provides less than 8% performance for 16 and 32 disk configurations, less than 11% for 64 disk configurations and less than 21% for 128 disk configurations. This is not surprising given the streaming nature of the applications of interest.

To provide an indication of price/performance ratio for the architectures compared in this study, we estimated the prices of all three architectures for 64 node configurations. We estimated the price of the SMP configuration using the SGI Origin 2000. The Avalon project at Los Alamos Labs quote the list price of a 64-processor SGI Origin 2000 with 250MHz processors and 8 GB memory to be about $1.8 million [8]. We estimated the price of the cluster configuration using the Micron PC ClientPro as the individual nodes, the Dell Poweredge 4300 as the front-end host, Seagate ST39102 as the disks and 3Com SuperStack II 3900 and SuperStack II 9300 as the Fast ethernet and gigabit ethernet switches. With these components, the price of a 64-node cluster adds up to about $167,000 ($1500 for each host, $670 for each disk, $300 for each network port and $9,000 for the front-end). We estimated the price of the Active Disk configuration using Seagate ST39102 as the disks with Cyrix 6x86 200MHz as the embedded processor, 32 MB SDRAM as the embedded RAM and $100 *premium* for being a high-end component. We assumed the same front-end host as for clusters with a LP3000 Fiber Channel host bus adaptor from Emulex Corporation[2]. With these components, the price for a 64 disk configuration adds up to about $74,000 ($1000 for each Active Disk, $600 for the Fiber Channel adaptor and $9000 for the front-end).

References

[1] A. Acharya and S. Setia. Availability and utility of idle memory in workstation clusters. Technical Report TRCS-98-26, Dept of Computer Science, University of California, Santa Barbara, Oct 1998.

[2] A. Acharya, M. Uysal, and J. Saltz. Active Disks: Programming Model, Algorithms and Evaluation. In *Proceedings of ASPLOS VIII*, pages 81–91, October 1998.

[3] A. Afework, M. Beynon, F. Bustamante, A. Demarzo, R. Ferriera, R. Miller, M. Silberman, J. Saltz, A. Sussman, and H. Tsang. Digital dynamic telepathology – the virtual microscope. In *Proceedings of the AMIA'98 Fall Symposium*, 1998. To appear.

[2] *http://www.emulex.com*

[4] S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proceedings of the 22nd International Conference on Very Large Databases*, pages 506–21, 1996.

[5] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD Conference on Management of Data*, pages 207–16, 1993.

[6] R. Agrawal and J. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–9, 1996.

[7] A. Arpaci-Dusseau, R. Arpaci-Dusseau, D. Culler, J. Hellerstein, and D. Patterson. High-performance sorting on networks of workstations. In *Proceedings of SIGMOD'97*, 1997.

[8] The Avalon FAQ[3], 1998.

[9] The Avalon Home Page[4], 1998. Avalon is a 140-processor Beowulf cluster at Los Alamos National Lab.

[10] E. Brewer, F. Chong, L. Liu, S. Sharma, and J. Kubiatowicz. Remote queues: Exposing message queues for optimization and atomicity. In *Proc. of the 7th SPAA*, pages 42–53, 1995.

[11] S. Donaldson, J. Hill, and D. Skillicorn. BSP Clusters: High Performance, Reliable and Very Low Cost. Technical Report PRG-TR-5-98, Oxford University Computing Laboratory, 1998.

[12] J. Eidenshink and J. Fenno. Source code for LAS, ADAPS and XID, 1995. Eros Data Center, Sioux Falls.

[13] G. Feldman. Source code for the SeaWIFS ocean data processing system, 1995. SeaWIFS group (NASA Goddard).

[14] R. Ferriera, B.Moon, J. Humphries, A. Sussman, J. Saltz, R. Miller, and A. Demarzo. The virtual microscope. In *Proceedings of the AMIA Fall'97 Symposium*, pages 449–53, 1997.

[15] G. Ganger, B. Worthington, and Y. Patt. The DiskSim Simulation Environment Version 1.0 Reference Manual[5]. Technical Report CSE-TR-358-98, Dept of Electrical Engineering and Computer Science, Feb 1998.

[16] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, Jun 1993.

[17] J. Gray. Some Challenges in Building Petabyte Data Stores. Distinguished Lecture, University of California, Santa Barbara, Oct 1997.

[18] J. Gray. What Happens When Processors Are Infinitely Fast and Storage Is Free? Keynote Speech at the Fifth Workshop on I/O in Parallel and Distributed Systems, Nov 1997.

[19] J. Gray. Put EVERYTHING in the Storage Device. Talk at NASD workshop on storage embedded computing[6], June 1998.

[20] J. Gray. The Sort Benchmark Home Page. Available at *http://research.microsoft.com/research/barc/-SortBenchmark/*, 1998.

[21] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proceedings of the 12th International Conference on Data Engineering*, pages 152–9, New Orleans, February 1996.

[3] http://cnls.lanl.gov/avalon/FAQ.html
[4] http://cnls.lanl.gov/avalon/
[5] Available at http://www.ece.cmu.edu/ ganger/disksim/disksim1.0.tar.gz
[6] *http://www.nsic.org/nasd/1998-jun/gray.pdf*

[22] D. Jiang and J. Singh. A methodology and an evaluation of the SGI Origin 2000. In *Proc. of the Intl. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 171–81, Madison, WI, June 1998.

[23] K. Keeton, D. Patterson, and J. Hellerstein. The Case for Intelligent Disks (IDISKS). *SIGMOD Record*, 27(3), 1998.

[24] J. Laudon and D. Lenoski. The SGI Origin: a ccNUMA highly scalable server. In *In Proc. of Intl. Symposium on Computer Architecture*, pages 241–51, Denver, CO, June 1997.

[25] L. McVoy and C. Staelin. lmbench: portable tools for performance analysis. In *In Proc. of 1996 USENIX Technical Conference*, Jan 1996.

[26] J. Melton and A. Simon. *Understanding the New SQL: A Complete Guide.* Morgan Kaufman, 1993.

[27] Mier Communications. Evaluation of 10/100 BaseT Switches. *http://www.mier.com/reports/cisco/-cisco2916mxl.pdf*, April 1998.

[28] G. Papadopolous. The future of computing. Unpublished talk at NOW Workshop, July 1997.

[29] D. Patterson et al. Intelligent RAM (IRAM): the Industrial Setting, Applications, and Architectures. In *Proceedings of the International Conference on Computer Design*, 1997.

[30] IBM Quest Data Mining Project. The Quest retail transaction data generator[7], 1996.

[31] E. Riedel, G. Gibson, and C. Faloutsos. Active storage for large scale data mining and multimedia applications. In *Proceedings of 24th Conference on Very Large Databases*, 1998. To appear.

[32] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–29, March 1994.

[33] Seagate Technology Inc. *The Cheetah 9LP Family: ST39102 Product Manual*, July 1998. Publication number 83329240 Rev B.

[34] P. Smith and B-B. Ding. Source code for the AVHRR Pathfinder system, 1995. Main program of the AVHRR Land Pathfinder effort (NASA Goddard).

[35] P. Strenstrom, E. Hagersten, D. Lilja, M. Martonosi, and M. Venugopal. Trends in shared memory multiprocessing. *IEEE Computer*, 1997.

[36] The 3Com SuperStack II Switch Datasheets. *http://www.3com.com/products/dsheets/400260a.html*, 1998.

[37] The Tolly Group. 3Com Corporation Superstack II 9300 Gigabit Ethernet Performance. Available off *http://www.tolly.com*, Jan 1998.

[38] M. Uysal, A. Acharya, R. Bennett, and J. Saltz. A customizable simulator for workstation clusters. In *Proc. of the 11th International Parallel Processing Symposium*, pages 249–54, 1997.

[39] R. Wahbe, S. Lucco, T. Anderson, and S. Graham. Efficient software-based fault isolation. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, pages 203–16, 1993.

[40] R. Winter and K. Auerbach. Giants walk the earth: the 1997 VLDB survey. *Database Programming and Design*, 10(9), Sep 1997.

[41] R. Winter and K. Auerbach. The big time: the 1998 VLDB survey. *Database Programming and Design*, 11(8), Aug 1998.

[42] M. Zaki, S. Parthasarathy, and W. Li. A localized algorithm for parallel association mining. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1997.

[7] Available at *http://www.almaden.ibm.com/cs/quest/syndata.html.*