Design and Evaluation of Smart Disk Architecture for DSS Commercial Workloads*

Gokhan Memik
ECE Department
Northwestern University
Evanston, IL 60208
memik@ece.nwu.edu

Mahmut T. Kandemir
CSE Department
The Pennsylvania State University
University Park, PA 16802
kandemir@cse.psu.edu

Alok Choudhary
ECE Department
Northwestern University
Evanston, IL 60208
choudhar@ece.nwu.edu

Abstract

The requirements for storage space and computational power of large-scale applications are increasing rapidly. Clusters seem to be the most attractive architecture for such applications, due to their low costs and high scalability. On the other hand, smart disk systems, with their large storage capacities and growing computational power are becoming increasingly popular. In this work, we compare the performance of these architectures with a single host-based system using representative queries from the Decision Support System (DSS) databases. We show how to implement individual database operations in the smart disk system and also show how to optimize the execution of the whole query by bundling frequently occurring operations together and executing the bundle in a single invocation. Besides decreasing the overall execution time, operation bundling also offers an easy-to-program and easy-to-use interface to access the data on smart disks. We also present a protocol for minimizing the communication time in the smart disk based system.

To measure the response times, we have developed the DBsim, an accurate simulator which can simulate the database operations for the single host-based, cluster-based and smart disk based systems. Using this simulator, we illustrate that the smart disk architecture offers substantial benefits in terms of overall query execution times of the TPC-D benchmark suite. In particular, the average response time of the smart disk architecture for the representative queries from the TPC-D benchmark in our base configuration is 71% smaller than the response time on the single host-based system and 4.2% smaller than the response time on the fastest cluster architecture. We also demonstrate the effectiveness of the operation bundling.

1. Introduction

The requirements for storage space and computational power of large-scale applications are increasing rapidly. Although SMP's and cluster of workstations offer high computational power, there is a need for new architectures especially for data-intensive applications. Such applications manipulate huge amounts of disk-resident data, in addition to their substantial computational requirements. In traditional systems, these data are moved back and forth between the storage device and the processing unit. This imposes an overhead on the I/O bus which may degrade the system performance. In the near future, the I/O interconnection is expected to become the bottleneck in the I/O subsystem due to the increases in the drive media rates.

For many of these large-scale applications, however, it is possible to manipulate data on the storage device, before putting it on the bus. *Smart disks*[1], having embedded processors and a substantial amount of memory on the storage device, solve this problem by manipulating the data on disks and leveraging the bandwidth requirement on the bus. In the near future, storage devices with 100 Gbytes of capacity, several hundred MIPS engine and a few hundred MBytes of RAM are expected to exist in the market [11]. Even today, it is possible to find storage devices in the market with 150 MIPS core and up to 2 MB of main memory [22, 24, 6, 14]. Most of the processing power in these disks is devoted to disk scheduling and similar duties. But, next generation smart disks will contain processors powerful enough for performing application-level programming. They might even contain co-processors for performing tasks related to disk scheduling. It is possible to build such systems with a small amount of extra cost over the disk cost due to the low costs of embedded processors and memory chips. Previous work in this area focuses on the architectural and operating system related issues [15, 20, 28]. Smart disks seems to be an attractive alternative especially for database applications. They are expected to perform well especially in a sequential operations, where significant amount of data has to be processed and to be sent to the processing unit, which usually performs a simple task on the data.

In this paper, we present a detailed quantitative evaluation of a smart disk based architecture. To achieve this, we compare the performances of a smart disk system, two types of cluster systems and a single host system for whole database queries.

Both SMP's and cluster of workstations are widely used for large scale applications. But clusters are getting increasingly popular due to their cost effectiveness. They are shown to perform well for many type of applications. Our goal is to measure the effectiveness of the emerging smart disk technology by comparing its performance to the existing popular technology of clusters.

We have selected Decision Support System (DSS) databases as our application, because of the large storage requirements and wide usage of such databases. Specifically, we measure the execution times, consisting of the I/O, computation and communication times, for all the architectures for six different queries from the TPC-D benchmark [25]. These queries contain a combination of select, join, sort, group-by and aggregate operations and are a representative of the whole benchmark suite. Our experiments show that, smart disk architecture delivers high levels of performance under different values for processor speed, available memory size, number of disks and database size. Based on our performance numbers, we also discuss the cases where the smart disk based system is preferable to cluster-based system and vice versa.

DSS databases process up to 4.5 TBytes of data, consisting of up to 50 billion rows [29]. These challenges require innovative approaches in archi-

*This work is supported by the Department of Energy's Accelerated Strategic Computing Initiative (ASCI) program under a subcontract No W-7405-ENG-48 from Lawarence Livermore National Laboratories and by NSF CDA-9703228 and NSF ACI-9707074.

[1] We use the term *smart disks* to refer to a class of architectures that put substantial computational power on disks, such as Active Disks [1, 20] and IDISKs [15].

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE

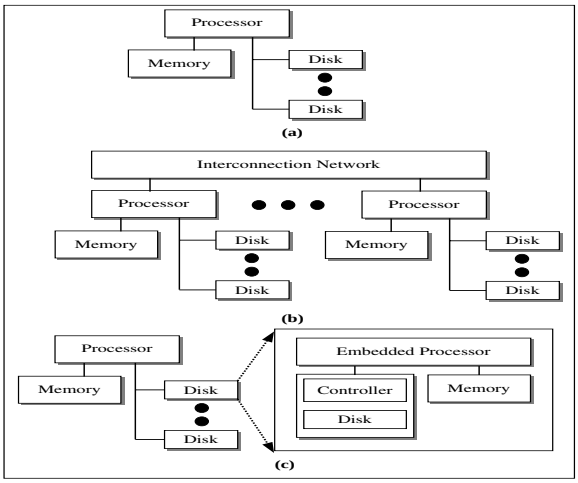

Figure 1. (a) Traditional architecture. (b) Cluster architecture. (c) Smart disk architecture.

tecture, software and algorithm areas because the traditional approaches, which depend on the technological advances for improving their throughput, may not be sufficient for solving these problems. Considering the results we have obtained in this work and the low cost of this architecture, the employment of smart disks in such applications seems to be an attractive solution.

In the following section, we describe the smart disk architecture and discuss the possible configurations of systems employing smart disks. In Section 3, we introduce the queries used in our experiments. In Section 4, we explain the algorithms we have used for single database operations and also explain the execution of the whole query using operation bundling. In Section 5, we present our simulator and discuss its accuracy. In Section 6, we describe our experimental platform, define the methodology used in the experiments, and present the simulation results. In Section 7, we discuss related work and in Section 8 we conclude the paper with a summary and an outline of the on-going research.

2 Smart Disk Architecture

Each smart disk consists of an embedded processor, a controller, disk space and some amount of DRAM (see Figure 1(c)). Compare this architecture with a traditional single host-based system (Figure 1(a)) and a cluster-based system (Figure 1(b)). In today's standards the CPU in Figure 1(a) is between $300-600$ MHz, with up to 1 GBytes of main memory. The I/O interconnect is between $200-300$ MB/s. To build a cluster, similar hosts are connected to each other using a fast-speed interconnection network. The speed of the interconnection is between $150-1200$ Mbps. We have simulated clusters with no shared disks. The embedded processor in Figure 1(c) is 100 to 300 MHz. As far as the memory is concerned, Texas Instruments C27x has a 16 MB address space [24]. So, we would expect memory sizes of 16 to 128 MB in the future.

Many alternatives for the cluster hardware exist, especially for larger systems. We have selected a configuration similar to Figure 1(b). Considering the size of our applications and the system size, we believe that our selection of the configuration is a reasonable one. The software issues for the cluster configuration is discussed in Section 3.

We can consider two alternatives for the configuration of smart disk system. In the first configuration, the smart disks are connected to a host machine through a bus. In such a system, host will do the tasks for security, coordination, code loading, etc. In such a system, smart disks will process the data and send only the relevant parts to the host (we call these *filtering* operations). But compute-intensive operations will still be performed by the more powerful host. This offloading of code performed by the smart disks does not only reduce the network traffic, but also offloads the host processor and increases the system power. The second alternative configuration is a distributed system of smart disks. In such a system, smart disks are connected through an interconnection network. One of the smart disks may be assigned as a central unit for coordination purposes, but all the applications are distributed among the smart disks. If parallelism in such a system can be exploited efficiently, systems with significant computing power and storage capacity can be constructed in a cost-efficient manner. The architecture we have used for our experiments falls into this category, with one of the smart disks assigned as a central unit.

For database applications significant amount of research has been conducted. First, there is literature on database machines, which were studied some time ago [4]. Special purpose hardware, which were employed by the database machines, had high cost and moderate performance, which eventually led the demise of database machines. Smart disk systems, on the other hand, use commodity hardware, lowering the cost of the system. Also the VLSI technology has improved dramatically, making smart disk systems feasible. The improvements in the interconnection network technology is also in favor of the smart disks. There has also been significant amount of research in parallel execution of database operations [7]. Considering each smart disk as a processing unit in the parallel database sense, we should be able to adapt at least some of these techniques to the smart disk architecture. Overall, armed with new optimization techniques from parallel databases and lessons from old database machines, we believe that we can build smart disk based systems which are cost-effective, practical and effective in handling large database applications.

3 DSS Queries

Dr. Philip Bernstein estimates that 35% of all database servers are decision support systems [5]. The storage and computational requirements of these systems increase rapidly. This wide usage and the large storage and computational requirements of these systems led us select them as our application in this work. We have used *six* queries from the TPC-D benchmark [25]. This benchmark has gained widely acceptance both in the academia and industry. It contains 17 read and 2 update queries, most of them being large and complex. The queries we have selected are given in Table 1 along with the operations they involve. A '$\times$' indicates that the query involves the relevant operation. For example, Q_1 involves S (sequential scan), sort, group-by and aggregate operations. We have selected these six queries, because we wanted to cover all the operations at least once.

The TPC-D benchmark gives the SQL codes for the queries. It defines some variables inside the query and also gives the possible values for these parameters. Thus, the possibility of a tuple being selected is fixed. For example, Q_{13} selects all the tuples from one of its input tables. On the other hand, Q_{12} selects one out of 200 tuples from a table called `lineitem`. Our choice of the queries also ensures that we experiment with both the low selectivity and high selectivity queries.

In the following, we first explain the implementation of individual database operations for both the smart disk and the cluster architectures. Then, we discuss how to combine these individual operations to execute the whole query. We introduce the notion of *operation bundling* and explain the protocol we devised for reducing the communication.

4 Query Execution

In this section, we first describe the algorithms we have used for individual database operations for all the architectures experimented with. Then, we explain how the whole query can be executed on smart disk system. We also explain the notion of operation bundling.

4.1 Individual Database Operations

Query optimization and processing in distributed environments has been studied by many researchers [12, 16, 23]. Many of the algorithms

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE

Table 1. The read-only TPC-D queries that we used and their operations. The operations are sequential scan (S), indexed scan (I), nested loop join (N), merge join (M), hash join (H), sort, group (Gro.), and aggregate (Agg.).

Query	Scan		Join			Sort	Gro.	Agg.
	S	I	N	M	H			
Q_1	×					×	×	×
Q_3		×	×			×	×	×
Q_6	×							×
Q_{12}	×	×		×		×	×	×
Q_{13}	×	×	×			×	×	×
Q_{16}		×			×	×	×	×

we have used in this work are adopted from the algorithms developed for distributed systems. We had to simplify some of the algorithms. But, these simplifications do not invalidate our comparisons, because we use the same assumptions and similar algorithms for both the cluster and the smart disk based architectures.

The implementations of individual database operations we have selected for smart disk architecture and clusters are similar in nature. The main difference of these architectures is the way these individual operations are combined to execute the whole query. These differences will be explained in Section 4.2 in more detail. The implementations of sequential scan, group-by and aggregate operations are similar to those proposed by Acharya et al. [2]. In the sequential scan operation, each smart disk scans the input table and sends the tuples that match the selection criterion to the central unit. Similarly, in the cluster architecture, hosts scan the input table and matching tuples are sent to the front-end, which concatenates the results. The aggregate operation is performed similarly. Each smart disk performs the aggregation locally and sends the results to the central unit, which combines the results. For indexed scan operation, we assumed that the smart disks keep the indexes for the part of the data they are holding. So, similar to the sequential scan, the smart disks scan their input table and forward the matching tuples to the central unit. The implementation is similar for the cluster architecture. For implementing the group-by operation, we have used a hashing based algorithm. In the first step, the local hashes are performed by each smart disk. Then, in the second step, these local hashes are sent to the central unit, which accumulates the results.

For the sort operation, we have used an external local sort in each disk. Then, these results are forwarded to the central unit (or to the front end), where the results are merged. Join operations require synchronization among the processing elements (smart disks in the smart disk system and hosts in the cluster architecture). For nested loop (N) join, one of the tables is replicated in all the processing elements. The selection for this table is done by the central unit in smart disk system. This table is joined with the local tables using a doubly nested loop to match the elements of one table to the other and the result is forwarded to the central unit (or to the front-end in cluster architecture). The merge (M) join starts by sorting one of the tables globally and replicating this sorted table in all the processing elements. Then, the local tables are merged with the global table and the results are forwarded to the central unit (to the front-end). For the hash (H) join, we first form the local hashes. Then, these hashes are communicated to form a global hash table. After receiving the global hash, the smart disks (the hosts) perform the join operation and the results are sent to the central unit (to the front-end).

4.2 Whole Query Execution

The execution of the whole query in smart disk architectures and cluster architectures differ in many ways. The processing elements in the clusters (hosts) are machines with their full operating system support and stand-alone database management systems. Their main difference from a single host-based machine is that they are aware of the other machines in the system and that each of them is setup to serve as an element in the whole system, which makes the whole system look as a single system to the clients. On the other hand, due to the limited memory and hardware, smart disks will not have the full support of the operating system or the database management system like their counterparts. Therefore, there must be a central unit in the system coordinating or synchronizing the operations of the smart disks in a finer grain. But, on the other hand, we believe that the smart disks will be powerful enough to control their memory and disk and will also be able to communicate with other smart disks without the intervention of the central unit.

The query execution on clusters is started by the front-end. Then, each host manipulates the data it owns. The hosts synchronize only when the operation they are performing requires the data on other machines. Among the individual operations we are performing, only the join operation requires such an synchronization. In other words, hosts perform the sequence of individual operations without any interruptions unless they encounter a join operation. If there is a join operation, they synchronize and proceed independently after the join operation is finished. Then, when all the operations are finished, they send their results to the front end.

In the following subsections, we are going to present the execution of the whole query in the smart disk architecture. We are going to define the notion of *operation bundling* and introduce a protocol we have devised for reducing the communication between the central unit and the smart disks, which in turn reduces the synchronization overhead.

4.2.1 Operation Bundling

The core of our approach for executing the whole query is to *bundle* a number of database operations and execute this bundle as a single operation on the smart disks. The execution of the bundles is coordinated by the central unit, which ensures that all the smart disks in the system are executing the same bundle.

The decision of which operations are to bundle is also made at the central unit. The algorithm uses a *relation of bindable operations* and the query plan tree as input. The relation of bindable operations consist of tuples of individual operations of the form *(child; parent)*. If there exists a (child; parent) tuple in the relation, this mean that any occurrence of these consecutive individual operations in the query plan tree should be included in the same bundle. The algorithm used for determining the bundles is given in Figure 2. It is a greedy algorithm for determining the bundles.

The execution of the query starts by forming a query plan tree [18]. Then, the tree is traversed starting from the root. When an individual operation is traversed, the algorithm checks all the children of the node. If a child and the parent are *bindable*, in other words if there exists a corresponding tuple in the relation (line 7 of Find_Bundle Algorithm), then the child is included in the current bundle and the algorithm continues recursively from the child. If they are not bindable, then the current bundle is finalized, a bundle containing the child is formed and the algorithm is called recursively with the child as the root of the plan tree. In summary, the algorithm traverses the entire query plan tree and bundles all the bindable individual operations. Figure 2 gives the algorithm of operation bundling. It uses a function called *insert* for both inserting a bundle to the list of finalized bundles and for inserting a node to a bundle. The algorithm returns a list of bundles. The corresponding bundles for $q12$ are given in Figure 3.

The success of the bundling algorithm depends heavily on the selection of the bindable individual operations. If this relation is empty, all the individual operations will be performed independently. If this relation contains all the possible combinations of tuples of individual operations, then the whole query plan tree will form a bundle. In this study, we have used a relation with the following tuples:

{*(indexed scan; nested loop join), (sequential scan; nested loop),*
(indexed scan, merge join), (sequential scan, merge join),
(indexed scan, hash join), (sequential scan, hash join),
(indexed scan, group-by), (sequential scan, group-by),
(group-by; aggregation)}

Many consecutive individual database operations can be bundled together to form the bundle. On one side, it is beneficial to bundle as many

```
FIND_BUNDLES (relation, root, current_bundle)
1. /*relation is used to store the
2. relation of bindable operations,
3. root is the root node of the
4. plan tree.*/
5. begin
6.   for i=0 to no.  of root's children do
7.     if relation[child_i, parent] == 1 then
8.       insert (child_i, current_bundle)
9.       FIND_BUNDLES (relation, child_i, current_bundle)
10.    elseif
11.      new_bundle = {}
12.      insert (child_i, new_bundle)
13.      FIND_BUNDLES (relation, child_i, new_bundle)
14.      insert (new_bundle, final_bundles)
15.    end if
16.  end for
17. return final_bundles
18. end.
```

Figure 2. Algorithm for determining the bundles.

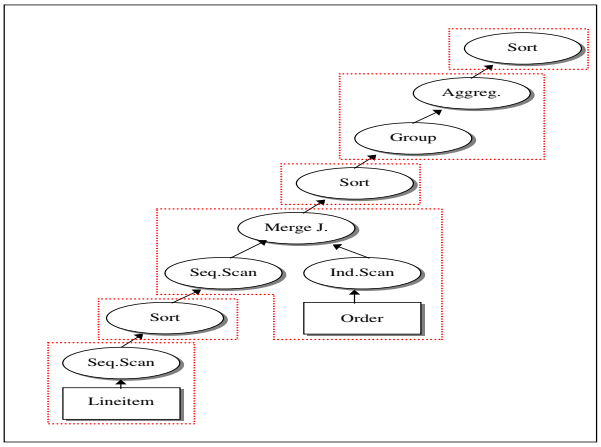

Figure 3. Query execution plan for Q_{12}. (Each dashed box contains a bundle).

operations as possible together, because this will reduce the amount of traffic and the synchronization overhead and also increase the performance. On the other side, having large number of single operations within a bundle will increase the number of bundles possible[2] and will require the smart disks to have more power and will make the query related system support more complex on the smart disk side. Our selection is based on the fact that, in all cases above, knowing the next individual operation can decrease the overall response time. Specifically, the tuples we have selected have at least one of the following properties:

- The results of the *child* operation can directly be supplied to the *parent* operation, thereby eliminating the need of storing the temporary results and also increasing the intra-query parallelism (for example, the results of a scan operation can directly be used by the join operation following it).
- The consecutive operations can be performed as a single operation, thereby decreasing the execution time (for example, while forming the groups the smart disks can also perform the aggregation operation).

In Section 6.2, we present experimental results with the above selection of relation of bindable operations and compare the results against no bundling. We also give results of another relation with more tuples and show that having additional tuples in the relation brings only marginal improvement.

The query execution starts on the central unit, where the query is parsed and optimized. These steps produce a query plan tree [18]. Then, the plan tree is fragmented by the central unit using the operation bundling algorithm described in this section. Then, the central unit sends each bundle to the smart disks and waits for its execution before sending the next one. The bundles are executed by the smart disks and the results are stored locally. Smart disks decide where to store the resulting data. According to the size of the produced data and of memory, the results are stored either in memory or on disk. In the last bundle, the central unit instructs the smart disks to send the result to the central unit. Then, it receives these results and combines them.

[2]The number of different possible bundles corresponds to the number of different sequences of individual operations that both satisfy the bindable operation relation and are encountered in at least one query plan tree. In other words, it corresponds to the number of different bundles the smart disk system should be able to execute for the set of queries supported.

5 Simulator

To conduct the experiments, we have developed a simulator, called DBsim, which is used to simulate the database operations for all the architectures. DBsim is capable of simulating both individual database operations and a sequence of individual operations. It can simulate a wide variety of disks, I/O interconnects and processors.

DBsim uses the Disksim developed by Ganger et al. [8], for simulating the disk behavior. Disksim is an efficient and accurate disk system simulator. It includes modules for simulating disks, intermediate controllers, buses, device drivers and request schedulers.

The sequential scan, indexed scan, sort, group-by, aggregate, nested loop join, merge join and hash join operations can be simulated in DBsim. DBsim is also capable of executing combinations of these individual operations. It requires both the architectural values like processor speed, memory size, disk parameters and also database related parameters like the total data size, location of data, index properties (e.g., whether there is an index on the attribute accessed or not, the type of index, etc.), tuple size, types and sizes of the resultant tuples and size of the resultant table.

The different architectures are simulated by using different programs driving the DBsim. The *single host simulator* is a sequential program, which reads the appropriate parameter values from a configuration file and calls DBsim with the appropriate arguments. The *cluster simulator* and the *smart disk simulator* are parallel programs. They read the parameter values from a configuration file and then they call the DBsim with the corresponding values. Then, according to the results obtained from DBsim, the processors communicate with each other using message passing.

To measure the accuracy of the DBsim, we have compared the response times of it against the values we have obtained from Postgres95 [30]. Postgres95 is installed on an IBM RS/6000 workstation with three IBMRISC DFHSS4W 4.5GB, 16-bit SCSI disk drive. We have measured the response times for the queries Q_3 and Q_6 from the TPC-D benchmark for two different database sizes and three different selectivities. The largest error DBsim has given was 2.4%. More information on DBsim and experiments we have conducted to validate the simulator is given in [17]. Overall, the DBsim simulator is found to be highly accurate.

6 Experiments

In this section, we present the simulation results obtained. First, we explain the base configuration used in the experiments. Then, we give

results for the experiments with different bundling schemes. Afterwards, the results for the *base configurations* of the single host-based, cluster-based and smart disk based systems are given. Finally, architectural and database values are changed to evaluate the potential performance of smart disks in the future. The results for all the variations are summarized in Table 3.

6.1 Parameters for Base Configuration

In this section, we explain the values used for architectural parameters in the base configuration. The host-based system has a CPU with frequency of 500 Mhz and has $256MB$ of main memory. The I/O interconnect is $200MB/s$. Each node in the cluster systems has a $400MHz$ CPU, $128MB$ of main memory and $200MB/s$ I/O interconnect. The nodes are connected via a $155Mbps$ inteconnect. Each smart disk has a CPU of $200MHz$ and $32MB$ of main memory. The disks employed by all the systems are of the same type. For all the systems experimented the total number of disks are kept constant at 8. The disks have a rotational speed of $10000rpms$, a minimum seek time of $1.62msec$, a mean seek time of $8.46msec$ and a maximum seek time of $21.77msec$. The data page size was $8KB$ for all the systems.

Our main goal was to have four systems of comparable prices. We also wanted the smart disk system to be the cheapest system to build. Although, it is hard to predict the market prices of smart disk systems, it is safe to assume that their manufacturing costs will be much less then the cluster or single-host based systems. One of the smart disks in the system is employed as the central unit, accomplishing the task of coordination.

6.2 Effect of Operation Bundling

To see the effect of the operation bundling, we have conducted an experiment with the base configuration explained in Section 6.1. We have experimented three different bundling schemes: *no-bundling*, bundling with the relation given in Section 4.2.1 (we call this scheme *optimal bundling*[3]), and *excessive bundling*. In no-bundling scheme, all the individual operations are performed independent of each other. Details about the optimal bundling are given in Section 4.2.1. For excessive bundling, we included the following tuples to the relation of bindable operations given in Section 4.2.1:

{*(indexed scan; sort), (sequential scan; sort), (sort; group-by), (sort; aggregate), (aggregate; sort), (aggregate; group-by)*}

Figure 4 gives the results obtained. The values in this figure correspond to the *percentage improvement of the overall execution time over the no-bundling scheme*. The average improvement over the no-bundling scheme is 4.98% with optimal bundling scheme and 4.99% with excessive bundling scheme. Note that, in Q_6, which consists of only two individual operations, no operations are bundled. Q_3 gives the best results among the queries we have examined. This query is one of the most complex queries and contains two join operations. It also produces significant amount of intermediate results. All these properties combined together, it has the best performance improvement among all the queries. The results show that building larger bundles does not improve the performance over the bundling scheme we have selected.

Overall, we can conclude that the operation bundling improves the performance of the smart disk systems. Another advantage of the communication using the operation bundling is the ease-of-programming. Using operation bundling, it is easy to program a portable, easy-to-use interface to access data residing on the disks.

6.3 Results for Base Configurations

In this section, we give the results for the configuration explained in Section 6.1. In each of the results (Figures 5 through 11) presented later in this section we compare four different architectures: *a traditional architecture with a conventional disk subsystem, a cluster consisting of 2 host machines, a cluster consisting of 4 host machines and a smart disk architecture*.

[3]This scheme may not give the optimal bundling for some queries

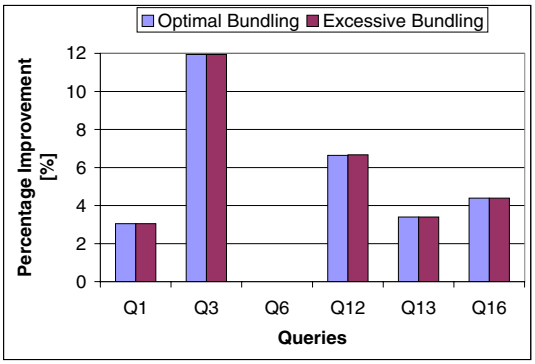

Figure 4. Results for operation bundling with the smart disk system having 8 disks.

Figure 5 presents the *normalized execution times* for the six queries for all the architectures. In this and the following figures the x-axis denotes the queries and y-axis denotes the execution times normalized with respect to the execution times of the single host-based system in base configuration. *The leftmost bar for each query shows the normalized time for the single host machine (i.e, it shows the new execution time divided by the execution time in base configuration), the second bar on the left represents the time for the cluster with 2 machines (i.e, it shows the new execution time of the cluster with 2 machines divided by the execution time of the single host-based machine in base configuration), the second right bar represents the time for the cluster with 4 machines, and the rightmost bar represents the relative execution time of the smart disk system*. Also, each bar for the host-based system is broken down into two components, the computation time and I/O time, whereas each bar for the clusters and the smart disk system is divided into three parts, the computation time, I/O time, and communication time. Computation time here denotes the time spent by all processors (host and smart) during the execution of the query code. The I/O time, on the other hand, is the time spent in I/O by host (in the traditional host-based system and in the clusters) and by smart disks (in the smart disk system). The communication time in clusters is the time spent in communication between the hosts. The communication time in smart disk system is the time spent in communication between the smart disks and the time spent in communication between the central unit and the smart disks.

The results show that in the base configuration the smart disk system has a speed-up between 2.24 and 6.06 for different queries, averaging 3.5 against the single host system. The smart disk architecture performed 43% better than the cluster with 2 machines and 4.2% better than the cluster with 4 machines in average. Note that, the cluster with 4 machines has twice as much memory with respect to all other systems. Only in Q_{16}, the cluster performed better than the smart disk system. And in Q_1, the cluster with 4 machines catch the performance of the smart disk system. Q_1 does not involve any join operation, which allows hosts in the cluster system to work independent from each other until the execution is finished. It also has a low I/O percentage. Because of these two reasons the cluster with 4 machines can perform as good as the smart disk system. Q_{16}, on the other hand, involves a hash based join operation. This operation requires substantial amount of main memory and computation. Therefore, cluster with 4 machines having larger total memory than the smart disk system favor from this property, resulting in a faster response time.

In conclusion, the smart disk architecture powered with fast serial links can bring significant amount of improvement in execution times over a host-based and can match the performance of more expensive cluster systems.

6.4 Sensitivity Analysis

In this section, we present the results obtained for different values of several architectural and database parameters. Table 2 shows all the varia-

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE

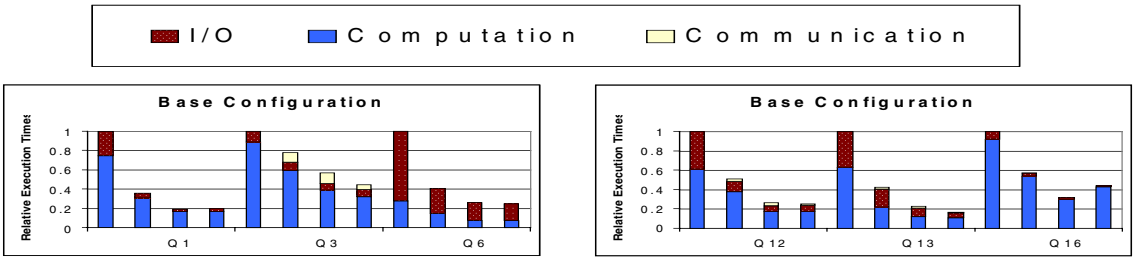

Figure 5. Relative execution times for the default configuration.

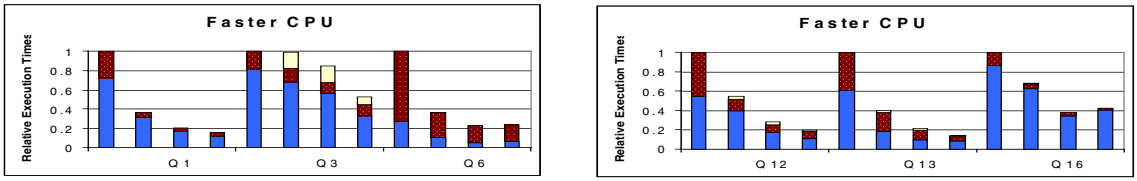

Figure 6. Relative execution times for faster CPU.

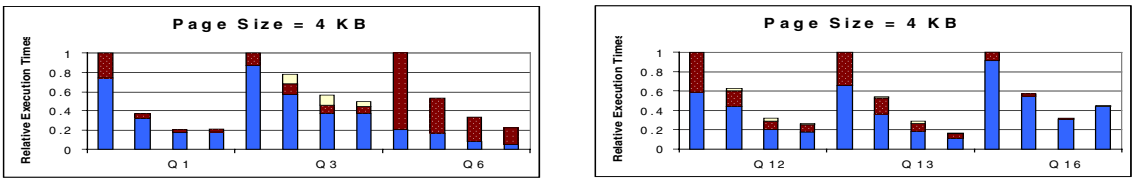

Figure 7. Relative execution times for smaller page size.

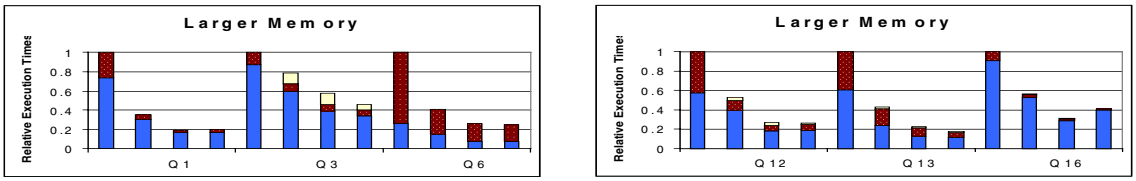

Figure 8. Relative execution times for larger memories.

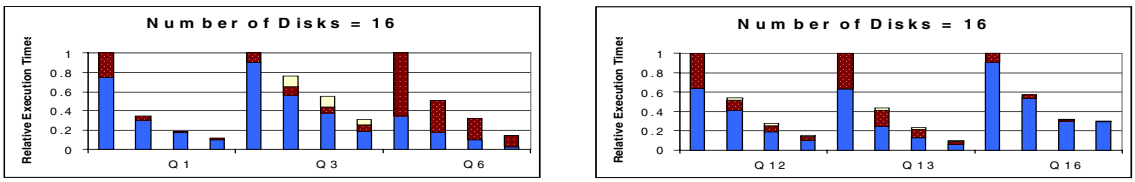

Figure 9. Relative execution times for more disks

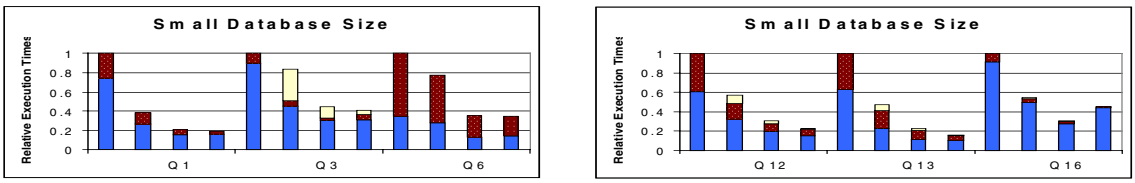

Figure 10. Relative execution times for smaller database size.

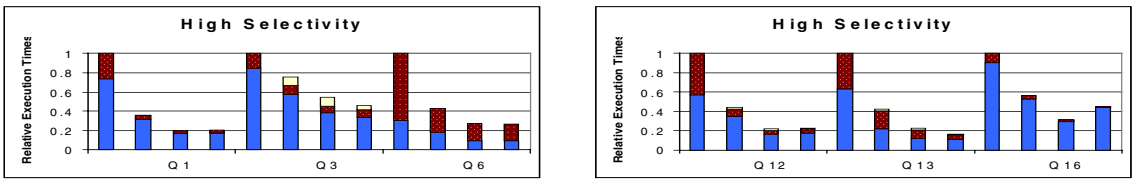

Figure 11. Relative execution times for higher selectivity

Four bars for each query represent the following architectures from left to right: single host based, cluster with 2 machines, cluster with 4 machines and smart-disk based systems.

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE

tions we have experimented with. It gives the new values for the variables whose values are modified for all the architectures. In each case, all the parameters except the relevant parameter remain the same as in the base configurations. The `small` database size corresponds to a database created using a scale factor[4] of $s = 3$. The scale factors for sizes `medium` and `large` are $s = 10$ and $s = 30$, respectively. The exact interpretations for the `small`, `medium`, and `large` database sizes and selectivities can be found in [17].

6.4.1 Varying Architectural Parameters

We start our architectural variations by increasing the CPU speed. Figure 6 reveals that increasing CPU speed increases the effectiveness of the smart disk system. The speed-up for the smart disk architecture increases to 3.56. The smart disk architecture performs 49.64% better than the cluster with 2 machines and 6.73% better than the cluster with 4 machines. Although the improvements in most of the queries are slight, the relatively larger improvement in Q_{16} resulted in an improvement in the average.

Next, we modified the data page size to see its effect on the performance. Due to space limitations, we only give the results for smaller page size. The averages for larger page size are given in Table 3. Figures 7 gives the results for the experiments with page sizes of 4 KB. Reducing the page size, reduces the effectiveness of the smart disk system. This is due to the fact that as the page size is increased, the size of the 'irrelevant' (unwanted) data increases, resulting in higher loads on the I/O bus.

When the memory sizes of all the architectures are doubled, the percentage decrease of the response times for all the architectures are similar. So, the relative performances remain as in the base configurations. The results for the experiment with larger memory sizes are given in Figure 8.

Finally, we changed the number of disks in all the systems, without changing the number of machines in cluster systems. Note that, as the number of disks is reduced in the smart disk system, the total computational power also drops and as the number of disks increases, the total computational power automatically increases. Figures 9 gives the results obtained for experiments with number of disks increased. The smart disk system has a speed-up of 5.38 when there are 16 disks in the system, showing that adding more disks to the single host machine (similarly to the hosts in the clusters) without increasing the computational power does hardly make a difference on the throughput of the system.

We have also conducted experiments with a faster I/O interconnection. The averages are given in Table 3. The individual results for each query is omitted due to space limitations.

As the technology advances, CPU speeds, memory sizes and number of disks in almost all the systems are increasing. Looking at the results we have obtained from our experiments, we can see that the smart disk architecture performs better with the increased CPU speed and the number of disks, and the performance improvement in smart disk system with the large memory size matches the increases for both the cluster systems and single host-based system. According to these results, we can conclude that technology advances will favor smart disk systems more than it does clusters and single host-based systems.

6.4.2 Varying Database Parameters

In this section, we present the experiments conducted with different database parameters. First, we experimented with different database sizes. Figure 10 give the results for smaller database size. For smaller database size, the speed-up drops to 3.32, which is matched by the cluster with 4 machines. The smart disk architecture performs better with larger database size, because as the size is increased, constant overheads of the smart disk system (synchronization, start-up, etc.) become negligable. As expected, increasing selectivity (Figure 11) decreases the effectiveness of the smart disk system. One of the advantages of the smart disk system is that the irrelevant data (e.g., database tuples which do not have any effect on the result of the operation) are not sent through the I/O bus, which enhances the I/O performance of the system. When the selectivity is increased, the proportion of this irrelevant data decreases, decreasing the advantage of the smart disk system.

[4] Scale factor corresponds to the total database size in GB; for instance, $s = k$ means that the total size of all the tables in the TPC-D database is k GB.

Table 3. Averages of experiments for different architectural and database related parameters (Each number corresponds to the average of the response times with respect to the single host machine for all queries).

Variation	*Single Host Mach.*	*Cluster with 2 Mach.*	*Cluster with 4 Mach.*	*Smart Disk System*
Base Conf.	100	50.6	30.3	29.0
Faster CPU	100	55.8	36.0	28.1
Large Page Size	100	48.6	29.2	25.6
Small Page Size	100	57.1	33.8	30.0
Large Memory	100	51.1	30.7	29.1
Faster I/O inter.	100	48.1	28.9	30.6
Fewer Disks	100	52.9	32.0	52.3
More Disks	100	50.1	29.6	18.6
Smaller DB. Size	100	59.7	30.1	30.1
Larger DB. Size	100	49.6	29.1	25.6
High Selectivity	100	49.3	29.5	29.4
Low Selectivity	100	52.3	31.5	28.5

7 Related Work

In the late 1970s and early 1980s there were numerous proposals on putting processing power in storage sub-systems. As an example, in the IBM 360, the I/O processors were able to execute *channel programs* that perform I/O on behalf of their hosts. Database machines employed processors on different levels of the disk architecture. For example, Banerjee et al. [4] proposed putting a processor per disk head. The others offered processor per track and processor per disk. As we have mentioned in Section 2, there are many differences between our work and the database machines. First of all, smart disk systems use commodity hardware, which makes them cost-effective. Secondly, we take efficient query optimization techniques into account and we also have experience in parallel databases.

Derived Virtual Devices [27] are proposed as a means of exporting devices with different capabilities to different users. DVDs are assumed to have IP-connectivity and that they can be accessed through a wide-area-network. This general-purpose connectivity increases the overhead of accesses to devices, specially when only communication between disk-processors and central unit is needed. The main focus of this work is to design mechanisms for maintaining security at the disks. Secure devices and migration of file-system capability to devices are also investigated as part of network-attached secure-disks (NASD) at CMU [9].

Acharya et al. [1] have recently proposed an Active Disk architecture which integrates significant processing power and memory into a disk drive and down-loads application specific code on disk. They hand-optimized a number of isolated database operators to run on the active disk architecture [2]. They also compare the performance of smart disk architecture, SMP's and clusters for a subset of individual database operations we have used in this work [2, 26]. Our research is different from theirs in the sense that we evaluated the execution of whole queries. Since a typical database query might involve parts that are suitable for the smart disks and parts that are not, it is very important to focus on the entire queries to reach a reliable evaluation. We have also implemented individual operations like indexed scan and hash join which were not considered in their work. We also use a different communication scheme. Riedel et al. [20, 19] have also focused on the active disk architecture and using them on database applications. Their work concentrates mainly on applications with almost no communication between disks. We believe that with the advances in serial communication links the disk-processors will be able to communicate with each other without the involvement of the hosts. Patterson et al. [15] present a disk architecture called intelligent disks (IDISKs) that puts processing power at the disks to overcome the I/O bus bottleneck of conventional systems. The main idea, as in active disks [1], is to off-load

Table 2. Variations in simulation parameters with respect to the base configuration.

Variations →	CPU $speed$	$Page$ $size$	$Memory$ $size$	I/O $inter.$	$Number$ $of Disks$	$Database$ $Size$	$Selectivity$
Single Host	1GHz	{4K,16K}	512MB	400MB/s	{4,16}	{small,large}	{small,large}
Clusters	800MHz	{4K,16K}	256MB	400MB/s	{4,16}	{small,large}	{small,large}
Smart Disk	350MHz	{4K,16K}	64MB	–	{4,16}	{small,large}	{small,large}

computation from expensive desktop processors. As compared to the active disks, the IDISKs are meant to be more general purpose.

8 Conclusions

Putting excessive computational power to the embedded systems and bringing processing units closer to the data are growing trends in computer architecture. Smart disks are a continuation of these trends. A smart disk system takes advantage of the processing power on disks by offloading user-defined code to the disks. This reduces the traffic in the I/O network, thereby increasing the system throughput dramatically. We evaluated a single host-based system, two cluster systems and a smart disk system using queries from the TPC-D benchmark. Our results show that the smart disk system can bring significant speed-ups for Decision Support System (DSS) databases. Specifically, for our base configuration, the smart disk system performed 71% better than the single host system, 43% better than the cluster with 2 machines and 4.2% better than the cluster with 4 machines. CPU speeds, memory sizes, total number of disks in the computer systems and the storage requirements of applications are increasing as the technology advances. Our experiments show that, for all these parameters, the relative performance of smart disk system increases as the parameter values are increased, except for the case of memory where the increase of performance of other systems is matched by the smart disk system. These performance results along with the cost-effectiveness of the smart disk systems make them very attractive for data-intensive applications. The work in-progress includes the design and implementation of automatic query optimizers that can handle other types of queries as well (e.g., update queries) and investigation of different applications for smart disk architecture.

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active disks: programming model, algorithms, and evaluation. In *Proc. ASPLOS VIII*, October 1998, pp. 81–91.

[2] A. Acharya, M. Uysal, and J. Saltz. Structure and performance of decision support algorithms on active disks. *Technical Report TRCS98–28,* Dept. of Computer Science, UCSB, October 1998.

[3] R. H. Arpaci-Dusseau, Eric Anderson, Noah Treuhaft, David E. Culler, Joseph M. Hellerstein, David Patterson, and Katherine Yelick. Cluster I/O with River: Making the Fast Case Common, *Input/Output for Parallel and Distributed Systems*, May, 1999.

[4] J. Banerjee, et al. DBC – A database computer for very large databases. *IEEE Transactions on Computers*, June 1979.

[5] P. Bernstein. Database Technology: What's Coming Next?, Keynote Presentation at *Fourth Symposium on High Performance Computer Architecture*, February 1998.

[6] Cirrus Logic, Inc. *Preliminary Product Bulletin CL-SH8665*, June 1998.

[7] D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. *Communications of the ACM,* June 1992, Vol.35, No. 6, pp. 85–98.

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

[9] G. A. Gibson et al. File server scaling with network-attached secure disks. In *Proc. of the ACM Sigmetrics,* Jun. 1997.

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

[11] J. Gray. Put everything in the storage device. Talk at *NASD Workshop on Storage Embedded Computing*, June 1998.

[12] W. Hasan and R. Motwani. Coloring Away Communication in Parallel Query Optimization. In *Proc. 21st Conference on Very Large Databases (VLDB'95),* 1995.

[13] High Performance Computing and Communications: Grand Challenges 1993 Report. *A Report by the Committee on Physical,Mathematical and Engineering Sciences*, Federal Coordinating Council for Science, Engineering and Technology, 1993.

[14] Intel Corporation. *i960 Hx Microprocessor Developer's Manual*, September 1998, Order Number:272484-002, Intel, Santa Clara, CA.

[15] K. Keeton, D. A. Patterson, and J. M. Hellerstein. The case for intelligent disks (IDISKs). In *SIGMOD Record*, 27(3), 1998.

[16] M. Mehta and D.J. DeWitt. Managing Intra-Operator Parallelism in Parallel Database Systems. In *Proc. 21st Conference on Very Large Databases (VLDB'95),* pp. 382-394, 1995.

[17] G. Memik, M. Kandemir, and A. Choudhary. An Experimental Evaluation of Smart Disk Architectures Using DSS Commercial Workloads. Technical Report CPDC-TR-9909-015, Dept of Electrical and Computer Engineering, Sep 1999.

[18] R. Ramakrishnan. *Database Management Systems,* The McGraw-Hill Companies, Inc., 1998.

[19] E. Riedel and G. Gibson. Active disks – Remote execution for network-attached storage. *Technical Report CMU-CS-97-198,*, School of Computer Science, Carnegie Mellon University, PA, 1997.

[20] E. Riedel, G. Gibson, and C. Faloutsos. Active storage for large scale data mining and multimedia applications. In *Proc. 24th Conference on Very Large Databases (VLDB'98),* New York, NY, 1998.

[21] C. Ruemmler, J. Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer*, Vol.27, No.3, March 1994, pp. 17–28

[22] Siemens Microelelectronics, Inc.*TriCore Architecture Overview Handbook*, February 1999.

[23] M. Stonebraker, et al. Mariposa: A wide-area distributed database system. In *VLDB Journal 5*, pp. 48-63, January 1996.

[24] A. Tessardo. TMS320C27x: New generation of embedded processor looks like a microcontroller, runs like a DSP. White Paper: SPRA446, Digital Signal Processing Solutions, March 1998.

[25] Transaction Processing Performance Council. *TPC Benchmark D Standard Specification Revision 2.1,* February 1998.

[26] M. Uysal, A. Acharya, and J. Saltz. Evaluation of Active Disks for Decision Support Databases. To appear in *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, Toulouse, France, January 10-12, 2000

[27] R. Van Meter, S. Hotz, and G. Finn. Derived Virtual Devices: A secure distributed file system mechanism. In *Proc. 5th NASA Conf. on Mass Storage Systems and Technologies*, Sept. 1996.

[28] R. Y. Wang, T. E. Anderson, and D. A. Patterson. Virtual log based file systems for a programmable Disk. *Proc. Third Symposium on Operating Systems Design and Implementation (OSDI'99),* February 1999.

[29] R. Winter and K. Auerbach. The big time: the 1998 VLDB Survey, *Database Programming and Design* Vol 11, No 8, August 1998

[30] A. Yu and J. Chen. The POSTGRES95 User Manual. Computer Science Div., Dept. of EECS, University of California at Berkeley, July 1995.