Journal of Parallel and Distributed Computing **61**, 1633–1664 (2001)
doi:10.1006/jpdc.2001.1743, available online at http://www.idealibrary.com on IDEAL®

Design and Evaluation of a Smart Disk Cluster for DSS Commercial Workloads

Gokhan Memik

Department of Electrical and Computer Engineering, Northwestern University, Evanston, Illinois 60208
E-mail: memik@ece.nwu.edu.

Mahmut T. Kandemir

Department of Computer Science and Engineering, Pennsylvania State University, University Park, Pennsylvania 16802
E-mail: kandemir@cse.psu.edu.

and

Alok Choudhary

Department of Electrical and Computer Engineering, Northwestern University, Evanston, Illinois 60208
E-mail: choudhar@ece.nwu.edu.

Received January 1, 2000; revised August 17, 2000; accepted October 4, 2000

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 and compare the scalabilities of the cluster-based and smart-disk-based systems. © 2001 Elsevier Science

Key Words: intelligent I/O; embedded systems; decision support systems; distributed databases; cluster of smart disks.

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 [16]. Even today, it is possible to find storage devices in the market with 150 Mips core and up to 2 MB of main memory [10, 22, 36, 39]. Intel's IQ-SDK system has a 2 MB memory, which can be expanded to 64 MB. Most of the processing power in these disks is devoted to disk scheduling and similar duaties. But, next generation smart disks will contain processors powerful enough for performing application-level programming. They might even contain coprocessors 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. Although this new type of smart-disk-based architecture seems very attractive, it poses many challenges. Previous work in this area focused on the architectural and operating system related issues [25, 34, 44]. Acharya *et al.* [1], on the other hand, focused on the implementation of the individual database operations. Smart disks seems to be an attractive alternative especially for database applications. Therefore, investigating the individual

[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, 34, 42] and IDISKs [25].

database operations on this architecture is very important. But, to gain more insight on the possible improvements by the smart disk architecture, we must examine whole queries from commercial workloads. Some individual operations may have good performance on smart disks and some may have poor performance. If the execution times or the occurrence frequency of the better operations are high, then the improvement with smart disks will be significant. If, on the other hand, such operations occur rarely, we will end up with a slight improvement. As an example, the smart disk architecture is expected to perform well in a sequential search operation, where all the data should be transferred to the processing unit. But for indexed selection, where an index structure is traversed and only the relevant tuples are brought to the processing unit, the improvements achieved by the smart-disk-based system may be lower because only the relevant data will be transfered from the storage unit to the processing element.

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. The main contributions of this paper are as follows:

- We present how a whole database query can be executed on a smart disk system.
- We present and evaluate a method called *operation bundling* for reducing the execution time of the database queries in smart disk architecture.
- We compare the execution times of whole database queries for single-host system, cluster system and smart disk system under several values of architectural parameters using an accurate simulator.

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 types 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 [41]. 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 the cluster-based system and vice versa.

DSS databases process up to 4.5 TBytes of data, consisting of up to 50 billion rows [45, 46]. These challenges require innovative approaches in architecture, 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 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 Fig. 1c). Compare this architecture with a traditional single host-based system (Fig. 1a) and a cluster-based system (Fig. 1b). In today's standards the CPU in Fig. 1a is between 300 and 600 MHz, with up to 1 GByte of main memory. The I/O interconnect is between 200 and 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 and 1200 Mbps. We have

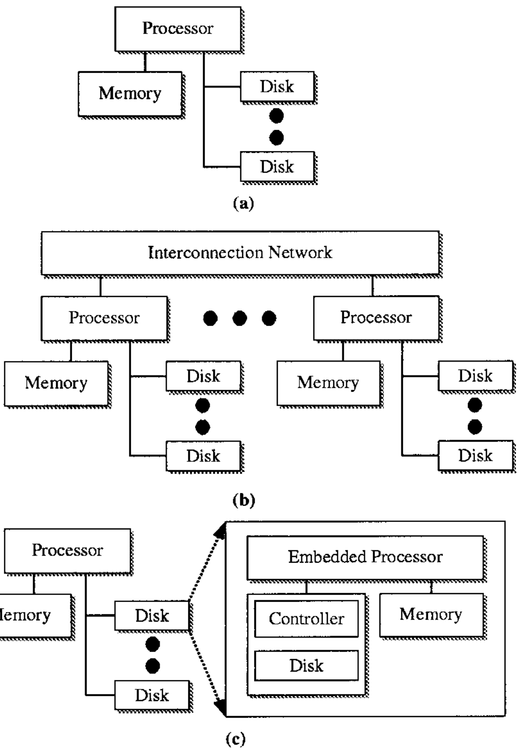

FIG. 1. (a) Traditional architecture. (b) Cluster architecture. (c) Smart disk architecture.

simulated clusters with no shared disks. Our selection of this configuration is based on the trends in the hardware for parallel databases. The embedded processor in Fig. 1c is 100 to 300 MHz. As far as the memory is concerned, Texas Instruments C27x has a 16 MB address space [39]. So, we would expect memory sizes of 16 to 128 MB in the future. According to Keeton *et al.* [25], the total cost of a smart disk system including fast serial links will be on the order of 10% of the total cost of a traditional host-based system.

Many alternatives for the cluster hardware exist, especially for larger systems. We have selected a configuration similar to Fig. 1b. 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 a smart disk system. In the first configuration, the smart disks are connected to a host machine through a bus. In such a system the host will carry out 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 because these operations discard the irrelevant data, resulting in a smaller amount of data transfer). But compute-intensive operations will still be performed by the host. In other words, the system will take advantage of the increasing computational power on disk devices by allowing filtering type of operations to operate on disks, close to data, and sending only the relevant information to the host. This offloading of code does not only reduce the network traffic, but also offloads the host processor and increases the system power. The emerging first generation of smart disk systems fall into this group [10, 22, 36, 39]. 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. Such a system will allow high performance computing for considerably low costs. The architecture we have used for our experiments falls into this category, with one of the smart disks assigned as a central unit. Using the first alternative would increase the cost of the system significantly, but would have increased the system performance only slightly, because of the high parallelism exhibited by the database applications.

There are also two important alternatives for programming of the smart disks. First, the smart disks can be programmed by the vendor. Second, the smart disks can upload user written code from the host machine. Both of these alternatives have advantages and disadvantages. Vendor implemented code will be compact, efficient, and easy to use, but the portability and flexibility will be important problems. There are several problems with the user written code, as well. We have assumed a vendor written programming model in our simulations. Therefore we assumed that most of the memory of the system can be used by the application.

Current technology is not in a position to employ smart disk systems for scientific codes. Even though parallelism is used widely in such applications, issues like code loading, language constructs, data partitioning among disks and the interaction between operating system and the disk resident code should be solved. Acharya *et al.* [1] propose a stream-based programming model for these purposes.

For database applications, however, a significant amount of research has already been conducted. First, there is literature on database machines, which were studied some time ago [6, 7, 26, 31]. Special purpose hardware, which was employed by the database machines, had high cost and moderate performance, which eventually led to 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. Unlike the old database machines, we do not have to perform all the operations on disks. We can make use of complex optimization schemes, most of which have been developed in the post-database machines era and can take the selectivity and cost of each operation into account and distribute the load in an efficient manner between the host and the smart disks. There has also been a significant amount of research in parallel execution of database operations [11]. 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 [8]. 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 to select them as our application in this work. We have used *six* queries from the TPC-D benchmark [41]. This benchmark has gained wide acceptance both in academia and industry. It contains 17 read and 2

TABLE 1

The Read-Only TPC-D Queries That We Used and Their Operations

	Select		Join					
Query	SS	IS	NL	M	H	Sort	Group	Agg.
Q_1	×					×	×	×
Q_3		×	×			×	×	×
Q_6	×							×
Q_{12}	×	×		×		×	×	×
Q_{13}	×	×	×			×	×	×
Q_{16}		×			×	×	×	×

Note. The operations are sequential scan (SS), indexed scan (IS), nested loop join (NL), merge join (M), hash join (H), sort, group, and aggregate (Agg.).

```
select
 l-shipmode,
 sum(case
  when o-orderpriority = '1-URGENT' or o-orderpriority = '2-HIGH'
  then 1
  else 0
 end) as high-line-count,
 sum(case
  when o-orderpriority <> '1-URGENT' and o-orderpriority <> '2-HIGH'
  then 1
  else 0
 end) as low-line-count
from
 orders,
 lineitem
where
 o-orderkey = l-orderkey
 and l-shipmode in ('[SHIPMODE1]', '[SHIPMODE2]')
 and l-commitdate < l-receiptdate
 and l-shipdate < l-commitdate
 and l-receiptdate ≥ date '[DATE]'
 and l-receiptdate < date '[DATE]' + interval '1' year
group by
 l-shipmode
order by
 l-shipmode;
```

FIG. 2. The SQL code for the query Q_{12} from TPC-D.

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 SS (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.

As an example, the SQL code of Q_{12} is given in Fig. 2. The possible values for the parameters (e.g., SHIPMODE1, SHIPMODE2, DATE) in the SQL codes are also defined by the benchmark. 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 the smart disk system. We also explain the notion of operation bundling.

4.1. Individual Database Operations

Query optimization and processing in distributed environments had been studied by many researchers [19, 20, 23, 28, 37]. Many of the algorithms 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 implementation 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. The central unit concatenates the tuples it receives from the smart disks. 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 and reports the results. For both of the architectures, we assumed that the data is distributed evenly on all the units in the system (except the central unit in the smart disk system, which performs only coordination and concatenation tasks). 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. Note that, for this operation more sophisticated sort algorithms, such as NOW-sort [3] can also be used. Join operations require synchronization among the processing elements (smart disks in the smart disk system and hosts in the cluster architecture). For nested loop (NL) 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 set up 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, 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 a 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. First, 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. Then, we are going to discuss the main idea with the help of an example.

4.2.1. Operation Bundling

The core of our approach for executing the whole query is to *bundle* (where appropriate) 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 at a time.

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 Algorithm 1. It is a greedy algorithm for determining the bundles. Although, it is not guaranteed to give the optimal result in all cases, for the queries we experimented with, it gives the optimal solution.

The execution of the query starts by forming a query plan tree [32]. 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 5 of Algorithm 1)[2] 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. Algorithm 1 gives the algorithm of operation bundling.[3] 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.

Algorithm 1 (Find_Bundles(*relation*, *root*, *current_bundle*)).

```
1.            /*relation is used to store the relation of bindable operations,
2.   root is the root node of the plan tree. */
3.   begin
4.     for i = 0 to number of children of root do
5.       if relation[child_i, parent] = = 1 then
6.         insert (child_i, current_bundle)
7.         FIND_BUNDLES (relation, child_i, current_bundle)
8.       elseif
9.         new_bundle = {}
10.        insert (child_i, new_bundle)
11.        FIND_BUNDLES (relation, child_i, new_bundle)
12.        insert (new_bundle, list_of_finalized_bundles)
13.      end if
14.    end for
15.    return list_of_finalized_bundles
16.  end.
```

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*),

[2] For representation of the relation, we have used a two diminsional matrix. Note that other representations, e.g., a list of tuples, can be used instead.

[3] This is a simplified version of the algorithm. In this version, the bundle containing the root of the tree is not included in the final list, but in the original algorithm this case is also considered.

(*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 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 possible number of bundles possible[4] and will require the smart disks to have more power and will make the query related system support more complex. Our selection is based on the fact that, in all cases above, knowing the next individual operation can decrease the overall response time. In other words, we have manually examined most of the basic database operations and selected all possible tuples that potentially increase the system performance. 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 having additional tuples in the relation brings only marginal improvement.

Table 2 summarizes the execution of the whole query and the communication protocol between the smart disks and the central unit. Query execution starts on the central unit, where the query is parsed and optimized. These steps produce a query plan tree [32]. 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 bundle. 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.

[4] 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. Note that the number of possible bundles and the average number of bundles in a query are inversely proportional. As the average number of individual operations within bundles increases, the number of different bundles which can be formed increases. But, since each bundle contains more individual operations, the average number of bundles satisfying a query decreases.

TABLE 2

Execution Protocol for the Smart Disk System

Task	Central unit	Smart disk architecture
`Query Parsing`	Parse query	—
`Query Optimization`	Perform query optimization	—
`Query Fragmentation`	Fragment query into bundles	—
	Send the bundle to the smart disks	Receive the bundle,
`Execution of the intermediate bundles`	Wait for the completion signal	Execute the bundle, Store the result, Send the completion signal to host.
	Send the bundle to the smart disk	Receive the bundle
`Execution of the last bundle`	Wait for the results	Execute the bundle, Send results to host.
`Combining the Results`	Combine the results of each smart disk	—

4.2.2. Example

In this section, we will explain the whole query execution by discussing the steps of executing the query Q_{12}. The SQL code for the query Q_{12} is given in Fig. 2 and the query execution plan is shown in Fig. 3 (this plan tree is from [40]), which also shows the bundles for the query.

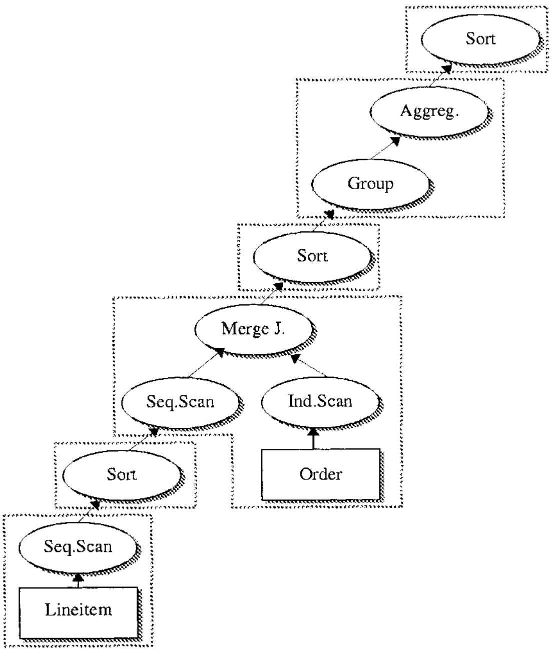

FIG. 3. Query execution plan for Q_{12}. (Each dashed box contains a bundle).

The execution starts on the central unit. Taking the SQL code, the central unit parses and optimizes the code and creates the execution tree. Then, the tree is fragmented using the algorithm described in Section 4.2.1. The resultant bundles are shown using the dashed boxes in Fig. 3. The algorithm starts by having the root operation *sort* as a bundle. Since the tuple (aggregation; sort) is not in the relation, it leaves the bundle as it is and continues from the aggregate node. Since there exists a tuple (group-by; aggregate), it includes the group-by operation to the bundle of aggregate operation. Since the next operation (sorting) cannot be added to this bundle (there exists no tuple (sort; group-by)), this is the final shape of the bundle. The algorithm keeps working its way in this fashion until all the nodes are visited and all the bundles are determined.

After determining the bundles, the central unit sends a message to the smart disks telling them to select the `lineitem.shipmode` and `lineitem.orderkey` fields of the table `lineitem` according to the parameters given in the query. This way, the selection and projection operations are performed as a single bundle. Receiving this message, smart disks scan the `lineitem` table sequentially. The smart disks store the results. If the resultant table fits into memory, it will be kept there, otherwise the data will be stored on disk. After finishing the operation, each smart disk sends a completion signal to the central unit. The central unit waits for all the smart disks to finish and then sends the next bundle to the smart disks. This new bundle instructs the smart disks to sort the resultant table from the previous bundle. When the execution of this bundle is finished, the next bundle is sent. This bundle contains a join and two select operations. The join operation requires one of the tables to be replicated in all the smart disks. The decision for the table to be replicated is done by the central unit and is indicated in the bundle message sent to the smart disks. The most important criterion in this selection is the size of the tables. For this query, it is more beneficial to take the `lineitem` table. Receiving the message, smart disks first replicate the filtered `lineitem` table by communicating each other. Then, each smart disk traverses this table and merges the matching elements from the local `order` table. Again the resultant table is stored in the smart disks. After this bundle, the central unit sends a message for sorting the resultant table. The next bundle contains the group-by and aggregate operations. In the last step, the central unit sends the instruction to sort the result table of the previous bundle. But this time, it indicates that the results should be sent to the central unit. Then, it waits for the incoming results, merges them, and finishes the execution of the whole query.

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.* [13], 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, and 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.

To simulate the communication times, DBsim performs the actual communication required by the query. This limits the number of different network configurations we can simulate, but gives a very accurate measurement of the communication overhead.

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 [47]. Postgres95 is installed on an IBM RS/6000 workstation with three IBMRISC DFHSS4W 4.5 GB, 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 database sizes we have experimented with have *scale factors*[5] of $s = 0.1$ and $s = 0.05$. The selectivities range from 2 to 4% and from 44 to 63% for Q_6 and Q_3, respectively.

The errors for changing the selectivity are 2.4 and 1.1% with the scale factor of $s = 0.1$ for the queries Q_3 and Q_6, respectively. The errors are calculated by dividing the difference of increase rates[6] in DBsim by the increase rates of Postgres95. The results were similar to the scale factor of $s = 0.05$. The errors for different database sizes are calculated in a similar way. The average of all the errors for different selectivities for changing the database size are 1.4 and 1.1% for the queries Q_3 and Q_6, respectively. The maximum error observed is 2.4%. More information on DBsim and experiments we have conducted to validate the simulator is given in [29]. Overall, the DBsim simulator is found to be highly accurate.

[5] 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 in k GB.

[6] Increase rate indicates the change in the execution time caused by a modification to the system parameters. We have used increase rates instead of absolute values to validate our simulator, because DBsim is a generic database simulator, it does not imitate any specific database management system. Therefore the increase rate is more important for us than the absolute execution times taken from any specific database management system.

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. Afterward, the results for the *base configurations* of the single host-based, cluster-based, and smart-disk-based systems are given. Then, 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 8. Finally, we compare the scalability of the smart disk architecture and the clusters.

6.1. Parameters for Base Configuration

In this section, we explain the values used for architectural parameters in the base configuration. Table 3 summarizes these values. We have selected the base configurations to represent existing state-of-the-art hardware. 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

TABLE 3

Base Configurations for the Experiments

Parameter	Value
Single host configuration	
Host CPU	500 MHz
Host memory	256 MB
I/O interconnect	200 MB/s
Configurationof clusters	
Host CPU	400 MHz
Host memory	128 MB
I/O interconnect	200 MB/s
Interconnect speed	155 Mbps
Smart disk configuration	
Disk CPU	200 MHz
Disk memory	32 MB
Disk-related values	
Number of disks	8
Rotation speed (rpms)	10000
Data page size	8 KB
Min. seek time	1.62 ms
Mean seek time	8.46 ms
Max. seek time	21.77 ms
Database-related values	
Size/relation	medium
Selectivity	medium

much less then the cluster or single-host based systems. In the base configuration, single host contains a 500 MHz CPU, 256 MB of RAM, and 8 disks connected through a 200 MB/s interconnection network. The hosts in both the clusters have a 400 MHz CPU with 128 MB RAM. For both of the cluster configurations, the total number of disks are kept constant at 8. So, the hosts in the cluster with 2 machines have 4 disks each and the hosts in the cluster with 4 machines have 2 disks each. The simulations for cluster systems are performed in a cluster of workstations connected through an ATM network with 155 Mbps peak bandwidth. Since the simulator performs the communication between the nodes, the simulated cluster system is also connected through an 155 Mbps ATM network. The smart disk system, on the other hand, consists of 8 smart disks, each having a 200 MHz CPU and 32 MB main memory. The simulations for the smart disk system were conducted on an IBM SP/2. So, the smart disks have fully connected TrailBlazer3 (TB3) switches between them with a 50.9 MBps peak bandwidth. Note that, neither the cluster system nor the smart disk employ the fastest available links. Networks with better performances can be used for both of the architectures, but this will increase the system cost dramatically. We wanted all the systems to have a comparable cost, having faster links for cluster and smart disk systems would violate this goal. If we employed faster links, this would have a minor effect on the relative performance of the cluster and the smart disk system. One of the smart disks in the system is employed as the central unit, accomplishing the task of coordination. The disks employed by all the systems are of the same type. The seek times along with the rotational speed of the disk are also given in Table 3.

6.2. Effect of Operation Bundling

To see the effect of the operation bundling, we have conducted two sets of experiments with the base configuration given in Table 3. We have experimented with a smart disk system having 4 disks, and also with a smart disk system having 8 disks. We have conducted experiments using three different bundling schemes: *no-bundling*, bundling with the relation given in Section 4.2.1 (we call this scheme *optimal bundling*), and *excessive bundling*. In no-bundling scheme, all the individual operations are performed independent of each other. For the six queries experimented in this study, this results in 8 different operations for the smart disks, each consisting one individual database operation. Optimal bundling results in 9 different possible bundles, each having 1.89 individual database operations in average. 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*)}[7]

[7] Note that these tuples do not correspond to different bundles. They are bindable individual operations. For more information about the relation of bindable operations see Section 4.2.1.

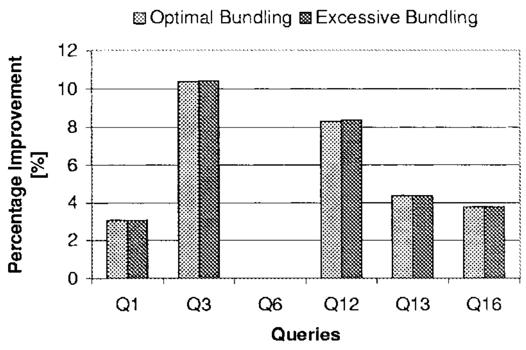

FIG. 4. Results for operation bundling with the smart disk system having 4 disks.

This has resulted in 11 different possible bundles,[8] each having 2.45 individual database operations in average. The number of different possible bundles and the average size of the bundles are an indication of the complexity of the system. The higher these numbers, the more complex the system is, because each different bundle possible should be supported by the smart disks.

Figures 4 and 5 give the results obtained. The values in these figures correspond to the *percentage improvement of the overall execution time over the no-bundling scheme*. Note that in Q_6, which consists of only two individual operations, no operations are bundled. Therefore the execution times for all bundling schemes are equal. 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. Three out of four bundles formed for this query contain more than one individual operation. It also produces a significant amount of intermediate results. All these properties combined together, it has the best performance improvement among all the queries. Overall, we can conclude that when the number of individual operations bundled increase, the execution time decrease.

The average improvement over the no-bundling scheme in the 4-disk system is 4.90% for both the optimal and excessive bundling schemes. The average improvement in the 8-disk system is 4.98% with optimal bundling scheme and 4.99% with excessive bundling scheme. On average, the percentage improvement increases with the number of disks increased. When the number of disks is increased, the duration of each individual operation decreases. Therefore, bundling consecutive operations result in longer uninterrupted code segments, thereby improving the effectiveness of the bundling. In some queries, however, the percentage improvement drops with the increased number of disks, because for no-bundling scheme larger percentage of the temporary results can be kept in memory with the increased number of disks.

These results show that building larger bundles does not improve the performance over the optimal bundling. There axe two reasons for this phenomena. First, we have included all the possible combinations of individual operations which would increase the intraquery parallelism; consecutively, building larger bundles

[8] Note that the actual number of different possible bundles is much higher than this number, but most of them are not formed in the queries we are experimenting with. For a discussion of the number of different possible bundles refer to Section 4.2.1.

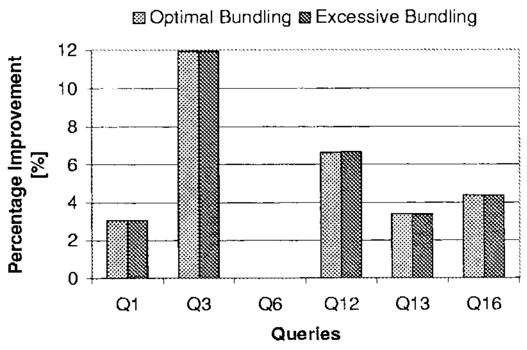

FIG. 5. Results for operation bundling with the smart disk system having 8 disks.

does not improve the parallelism within the query. Second, the synchronization cost is also not decreased with excessive bundling, because some of the operations require synchronization regardless of the size of the bundle. The number of synchronization points cannot be less than the number of such operations. Therefore, increasing the bundle size does not decrease the number of synchronization points significantly.

Overall, we can conclude that the operation bundling improves the performance of the smart disk system. 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

The base configurations are shown in Table 3. In each of the results (Figs. 6 through 17) 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.* The idea in conducting these experiments is not to show that the smart disk system performs better then the conventional systems, but our goal is to show that it is possible to build a cheaper and less complex system that performs as good as the available more costly systems.

Figure 6 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

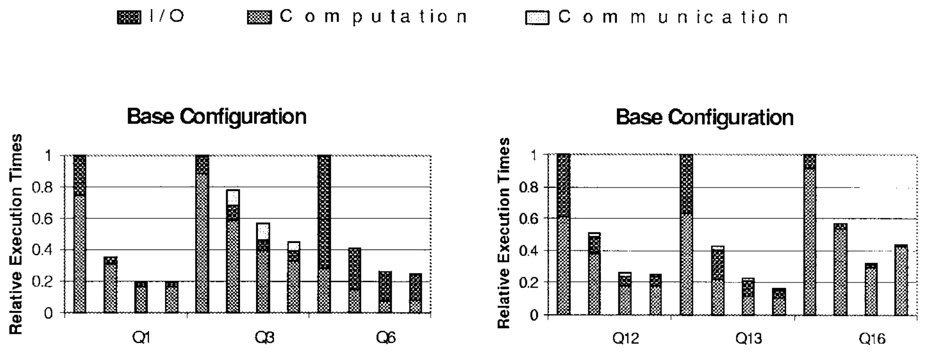

FIG. 6. Relative execution times for the default configuration.

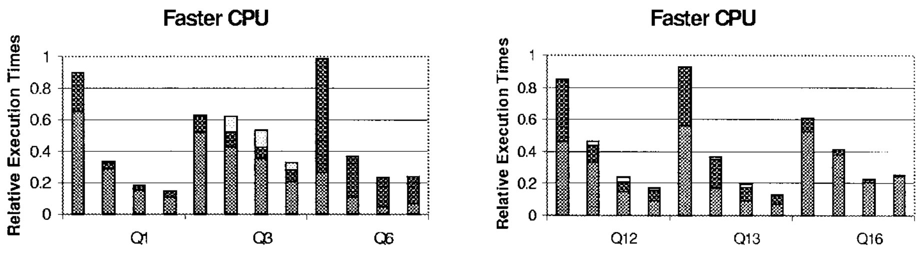

FIG. 7. Relative execution times for faster CPU.

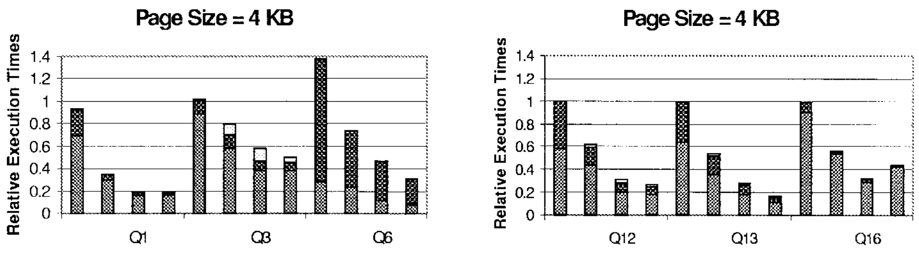

FIG. 8. Relative execution times for smaller page size.

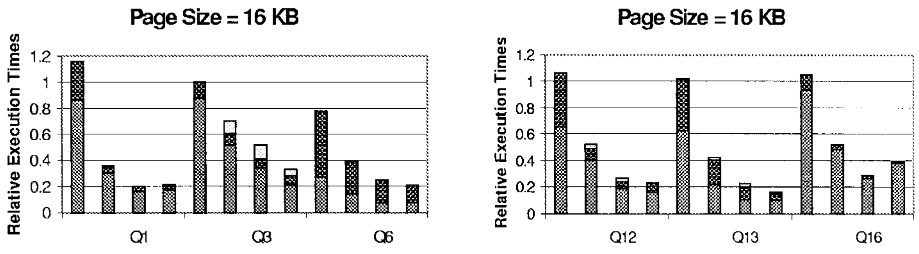

FIG. 9. Relative execution times for larger page size.

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 average speed-ups for the cluster with 2 machines and 4 machines are 1.96 and 3.30, respectively. The smart disk architecture performed 43% better than the cluster with 2 machines and 4.2% better than the cluster with 4 machines on average. Note that the cluster with 4 machius has twice as much memory with respect to all other systems. Only in Q_{16}, did the cluster perform better than the smart disk system. In Q_1, the cluster with 4 machines matches 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. Another property of this query is that it has a low I/O percentage, which allows the full utilization of the host processors. Because of these two reasons the cluster with 4 machines can perform as well as the smart disk system. Q_{16}, on the other hand, involves a hash based join operation. This operation requires a substantial amount of main memory and computation. Although constructing the global hash table can be parallelized easily, its building from the local hash tables takes a significant amount of time, for which the overhead increases with the number of processors involved. Therefore, a cluster with 4 machines having larger total memory than the smart disk system and the same computational power with less machines favor from these properties, resulting in a faster response time.

A very important property of all the experiments conducted in this work is that the communication time is relatively small compared to the total running time, especially for the smart disk architecture. This is due to the use of fast links employed and the protocol we have developed for the smart disk architecture. For all the experiments conducted, we have assumed a perfect data load balancing. This assumption, together with the low synchronization costs resulted in such high speed-ups for both the clusters and the smart disk architecture.

6.4. Sensitivity Analysis

In this section, we present the results obtained for different values of several architectural and database parameters. Table 4 shows all the variations with which we have experimented. 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 (Table 3). For instance, in the first variation, changing the CPU speed does not affect the memory size. The interpretations for the small, medium, and large database sizes and selectivities are given in Tables 5 and 6, respectively. Table 7, on the other hand,

TABLE 4

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	1 GHz	{4K, 16K}	512 MB	400 MB/s	{4,16}	{small, large}	{small, large}
Clusters	800 MHz	{4K, 16K}	256 MB	400 MB/s	{4,16}	{small, large}	{small, large}
Smart Disk	350 MHz	{4K, 16K}	64 MB	—	{4,16}	{small, large}	{small, large}

TABLE 5

Total Size of the Tables

	Q_1	Q_3	Q_6	Q_{12}	Q_{13}	Q_{16}
small	2.4GB	3GB	2.4GB	3GB	0.75GB	0.6GB
medium	8GB	10GB	8GB	10GB	2.5GB	2GB
large	24GB	30GB	24GB	30GB	7.5GB	6GB

TABLE 6

Selectivities of the Queries [%]

	Q_1	Q_3	Q_6	Q_{12}	Q_{13}	Q_{16}
small	83	44	2	0.4	100	12
medium	88	54	3	0.8	100	15
large	93	63	4	1.6	100	18

TABLE 7

Number of Tuples in the Accessed Tables (M denotes *Millions*)

	Q_1	Q_3	Q_6	Q_{12}	Q_{13}	Q_{16}
small	18M	18M × 4.5M × 0.45M	18M	18M × 4.5M	4.5M × 0.45M	2.4M × 0.6M × 30K
medium	60M	60M × 15M × 1.5M	60M	60M × 15M	15M × 1.5M	8M × 2M × 0.1M
large	180M	180M × 45M × 4.5M	180M	180M × 45M	45M × 4.5M	24M × 6M × 0.3M

gives the corresponding number of tuples in the database for different database sizes.

6.4.1. *Varying Architectural Parameters*

We start our architectural variations by increasing the CPU speed. Figure 7 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, whereas the speed-ups for both the cluster drop to 1.79 and 2.78 for clusters with 2 machines and 4 machines, respectively. So, 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. Figures 8 and 9 give the results for the experiments with page sizes of 4 and 16 KB, respectively. As the page size is increased, the effectiveness of the smart disk system increases. 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 Fig. 10. We see from the figures that the I/O times also drop when the memory size is increased for most of the queries.

Increasing the speed of the I/O interconnection, favors mostly for the cluster with 4 machines. Even though data can be transfered faster to the processing unit in the single host system, since the CPU is already highly utilized (for most of the queries), the throughput of the system does not change dramatically. On the other hand, for the cluster with the 4 machines, faster data retrieval can result in higher throughputs. The results for these experiments are plotted in Fig. 11. The speed-up for smart disk architecture drops to 3.27 and the cluster with 4 machines performs 5.56% better than the smart disk system for this configuration. In these experiments, we could not change the speed of the interconnection network for the smart disks, due to the hardware limitations.

Finally, we changed the number of disks in all the systems, without changing the number of machines in cluster systems (the experiments with different number machines are given in Section 6.5). 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 12 and 13 give the results obtained for experiments with different numbers of disks. The speed-up against the single host machine for the smart disk architecture with 4 disks drops to 1.91 on average, matching the speed-up of the cluster with 2 machines. The cluster with 4 machines, on the other hand, has a speed-up of 3.13. In contrast, 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 hardly makes a difference on the throughput of the system.

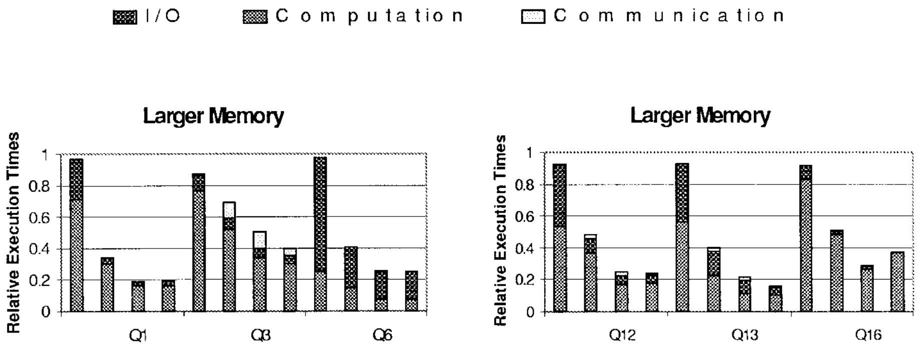

FIG. 10. Relative execution times for larger memories.

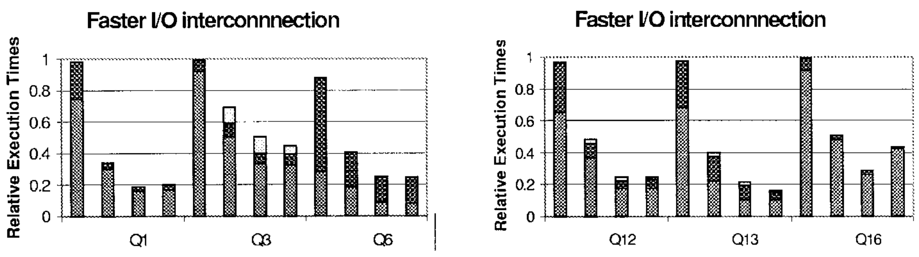

FIG. 11. Relative execution times for faster I/O interconnection.

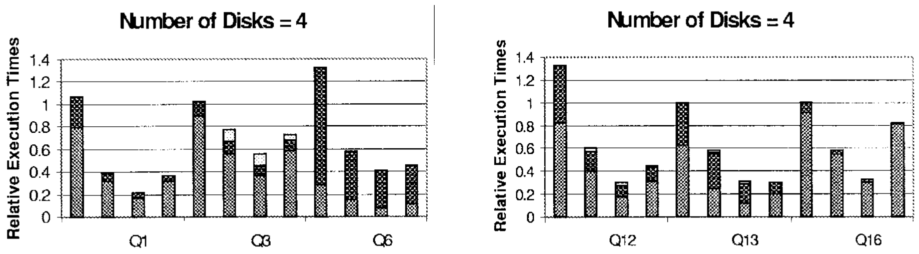

FIG. 12. Relative execution times for less disks (without increasing the number of hosts in the clusters).

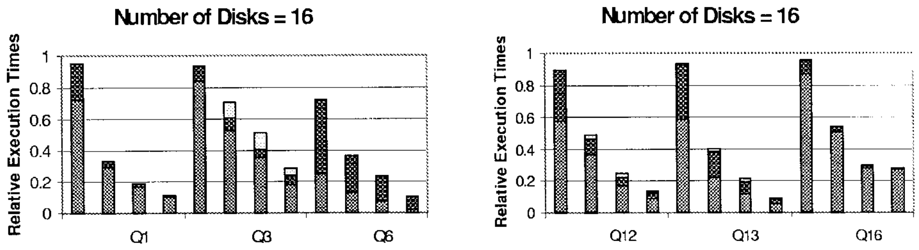

FIG. 13. Relative execution times for more disks (without increasing the number of hosts in the clusters).

As the technology advances, CPU speeds, memory sizes, and the 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 15 reveals that increasing the database size increases the performance of the smart disk system. The speed-up for the database with large size with the smart disk architecture is 3.91, which performed 12.03% better than the cluster with 4 machines and 48.39% better than the cluster with 2 machines. For smaller database size (Fig. 14), 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 (Fig. 16) decreases the effectiveness of the smart disk system and decreasing selectivity favors smart disk systems (Fig. 17). 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. This effect of selectivity is similar to the results obtained by Riedel *et al.* [34].

6.5. Scalability

In this section, we compare the scalability of clusters and the smart disk system. To measure the scalability, we conducted experiments with 4, 8, 16, and 32 disks. In these experiments, the number of disks per host remains the same, so we increased the number of hosts in the system to increase the total number of disks. We used the same type of clusters discussed in the previous sections, i.e., hosts with 2 disks and 4 disks each. Again as before, the hosts are connected through a 155 Mbps ATM network, with an ATM switch for each 4 cluster machine simulated. The configuration for the smart disk system and the values for the database related parameters are also as in the base configurations (Table 3).

Figures 18 through 20 give the results obtained from our experiments. For each query, the leftmost bar represents the speed-up of the cluster consisting of hosts with 4 disks, the middle bar represents the speed-up of the cluster consisting of hosts with 2 machines, and the rightmost bar represents the speed-up of the smart disk system. To calculate speed-ups, we take the response time of each architecture with 4 disks as the base. In other words, the speed-up is calculated by dividing the

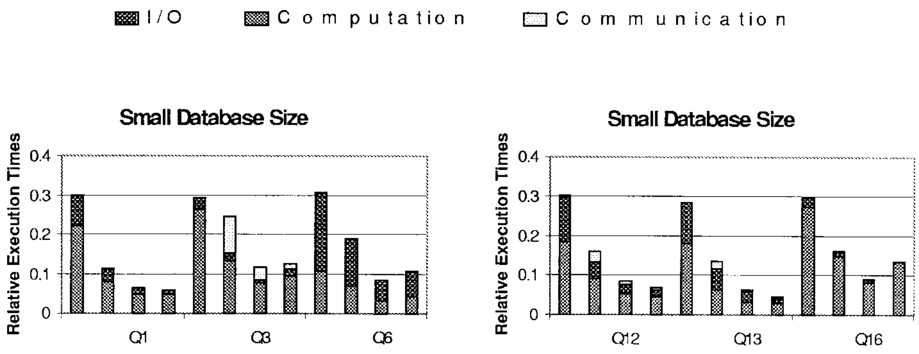

FIG. 14. Relative execution times for smaller database size.

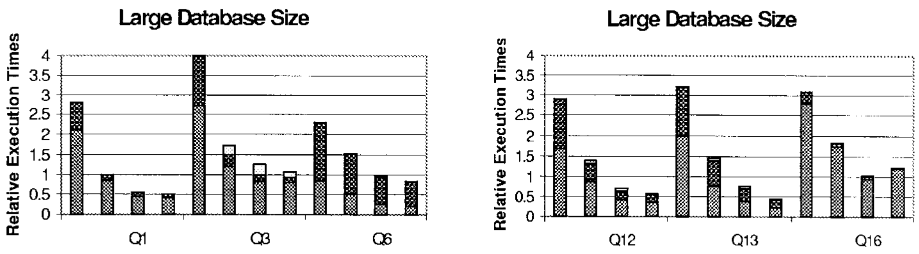

FIG. 15. Relative execution times for larger database size.

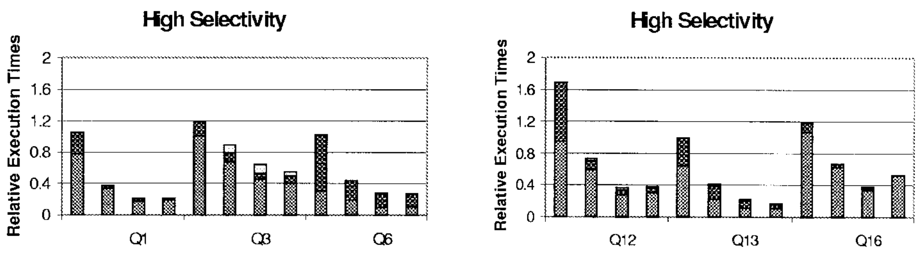

FIG. 16. Relative execution times for `high` selectivity.

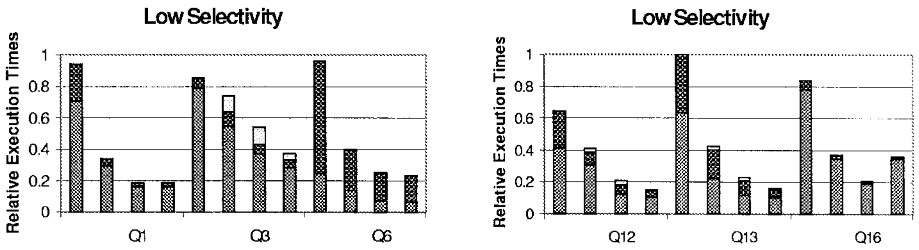

FIG. 17. Relative execution times for `low` selectivity.

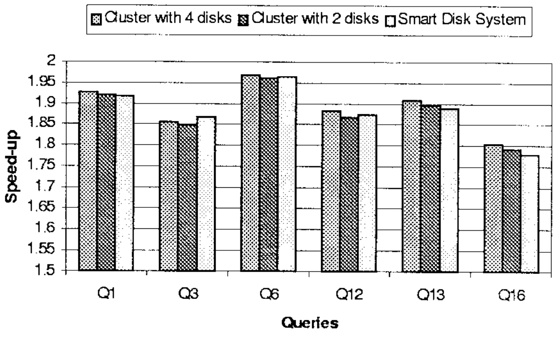

FIG. 18. Speed-ups of the three architectures with 8 disks for different queries.

corresponding response time of the architecture with 4 disks by the response time of the *same architecture* with the corresponding number of disks (8 disks for Figs. 18, 16 disks for Figs. 19, and 32 disks for Fig. 20). For all the figures, we can say that the scalabilities for all the architectures are similar except for the query Q_{16}. For this query, the speed-up of the smart disk architecture is slightly less than the speed-ups of clusters. The reason for this result is that this query produces relatively large results, which should be combined by the central unit. As the number of disks in the system is increased, combining the local results becomes a bottleneck. Clusters are less affected by this bottleneck, because they have a faster front-end. Note that the combination of the final result from the local results can also be performed in parallel, which will increase the speed-up for all the systems. Another solution to this problem may be to reconfigure the smart disk system. Instead of having one central unit for all the smart disks, smart disks can be collected in several groups, each with its central unit. Then, to increase the total number of disks in the system, all we have to do is to add more groups, each having a central unit. Such a configuration of smart disks will be more scalable than our current configuration. But considering the sizes of the systems we have experimented with (8 disks in default configuration), such a rearrangement of smart disks will have an insignificant effect on the results of the previous sections. The effect of a slower central unit can also be observed in queries Q_1 and Q_{13} to a lesser extent. On the other hand, for Q_3, the smart disk architecture scales better especially up to 16 disks. This

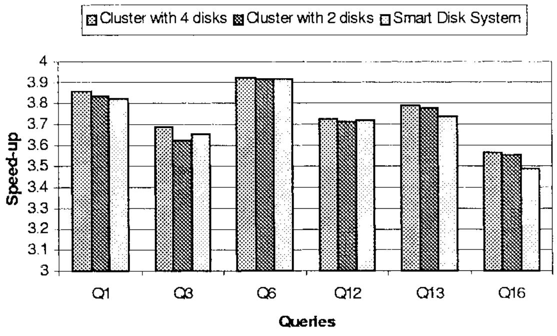

FIG. 19. Speed-ups of the three architectures with 16 disks for different queries.

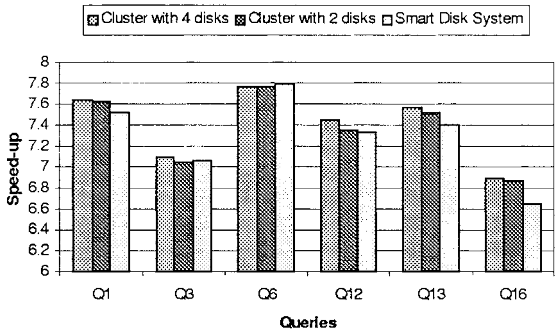

FIG. 20. Speed-ups of the three architectures with 32 disks for different queries.

TABLE 8

Averages of Experiments for Different Architectural and Database-Related Parameters

Variation	Single host machine	Cluster with 2 machines	Cluster with 4 machines	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	38.9	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

Note. Each number corresponds to the average of the response times with respect to the single host machine for all queries.

TABLE 9

Average Speed-Ups for Scalability Experiments

Number of disks	Cluster with 4 disks	Cluster with 2 disks	Smart disk system
8	1.89	1.88	1.88
16	3.75	3.73	3.72
32	7.39	7.36	7.31

Note. Each number corresponds to the average of the speed-ups for all the queries.

query is the query with the largest communication requirement. The communication overhead increases if the total number of machines in the system is increased. Having a smaller communication percentage, the smart disk system is less affected by this overhead. But as the number of disks is increased, the overhead of a single central unit becomes the dominant factor, favoring the clusters.

Table 9 gives the average speed-ups of all queries for all the architectures and for different numbers of disks. With a slight effort of putting in a faster central unit, the scalability of the smart disk system can be improved significantly. Also, the speeds of embedded processors are increasing rapidly, which will favor the smart disk architecture. Overall, our results show that the scalability of the smart disk system matches the scalability of clusters.

7. RELATED WORK

In the late 1970s and early 1980s there were numerous proposals for putting processing power in storage subsystems. 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.* [7] proposed putting a processor per disk head. The others offered processor per track and processor per disk [6, 26, 31]. Unfortunately, the special-purpose components eventually led to the demise of earlier database architectures. 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 [43] 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, especially 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 [14].

Acharya *et al.* [1] have recently proposed an Active Disk architecture which integrates significant processing power and memory into a disk drive and downloads 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, 42]. Their simulation results show that the processing power on disks might be useful. 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.* [33, 34] have also focused on the active disk architecture and have used it in database applications. Their work is concentrated 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.* [25] 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 computation from expensive desktop processors. Compared to the active disks, the IDISKs are meant to be more general purpose.

Virtual disks [5] and logical disks [24] provide a device abstraction that hide the physical connectivity of the disk. These systems make a disk connected to a remote host to appear as if it is logically/virtually connected to a local host. Striping-related work [12, 18, 27] studies the impact of device-level and server-level parallelism on I/O performance. These systems still treat disks merely as storage. Our work will extend the function of any disk on the network to encompass local processing at the disk.

Active Networks [30, 38] envisions code migration to the network to improve QOS. These projects [9, 17] deal with execution of user code (carried within network packets) in the network. The migratable code is much more application-specific in smart disk systems as opposed to dealing with a number of predefined QOS attributes of the network.

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 off-loading 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. We also showed that the the scalability of the smart disk system matches the scalability of clusters. 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 (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,” pp. 81–91, October 1998.
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. A. Arpaci-Dusseau, R. Arpaci-Dusseau, D. Hellerstein, and D. Patterson, High performance sorting on networks of workstations, *in* “Proc. SIGMOD’97,” 1997.
4. R. H. Arpaci-Dusseau, E. Anderson, N. Treuhaft, D. E. Culler, J. M. Hellerstein, D. Patterson, and K. Yelick, Cluster I/O with river: Making the fast case common, *Input/Output Parallel Distrib. Systems* (May 1999).
5. C. R. Atanasio, M. Butrico, C. A. Polyzois, S. E. Smith, and J. L. Peterson, “Design and Implementation of a Recoverable Virtual Shared Disk,” IBM Technical Report RC 19843, Nov. 1994.
6. E. Babb, Implementing a relational database by means of specialized hardware, *ACM Trans. Database Systems* **4**(1), March 1979.
7. J. Banerjee *et al.*, DBC—A database computer for very large databases, *IEEE Trans. Comput.* (June 1979).
8. P. Bernstein, Database technology: What’s coming next?, Keynote Presentation at, *in* “Fourth Symposium on High Performance Computer Architecture,” February 1998.
9. S. Bhattacharjee, K. Calvert, and E. W. Zegura, Implementation of a active networking architecture: White paper presented at “Gigabit Switch Technology Workshop,” Washington University, St. Louis, July 1996.
10. Cirrus Logic, Inc., *Preliminary Product Bulletin CL-SH8665*, June 1998.
11. D. DeWitt and J. Gray, Parallel database systems: The future of high performance database systems, *Comm. Assoc. Comput. Mach.* **35**(6) (June 1992), 85–98.
12. A. Drapeau *et al.*, RAID-II: A high-bandwith network file server, *in* “Proc. of 21st Ann. Symp. on Computer Architecture (ISCA’94),” pp. 234–244, April 1994.
13. 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.
14. G. A., Gibson *et al.*, File server scaling with network-attached secure disks, *in* “Proc. of the ACM Sigmetrics,” June 1997.
15. G. Graefe, Query evaluation techniques for large databases, *ACM Comput. Surveys* **25**(2) (June 1993), 73–170.
16. J. Gray, Put everything in the storage device, presented at “NASD Workshop on Storage Embedded Computing,” June 1998.
17. J. Hartman, U. Manber, L. Peterson, and T. Proebsting, “Liquid Software: A New Paradigm for Networked Systems,” Technical Report 96-11, Department of Computer Science, University of Arizona, 1996.
18. J. H. Hartman and J. K. Ousterhout, The Zebra striped network file system, *in* “Proc. of the 14th Symposium on Operating System Principles (SOSP’93),” pp. 29–43, Dec. 1993.
19. W. Hasan, “Optimization of SQL Queries for Parallel Machines,” Ph.D. thesis, Stanford University, 1995.
20. W. Hasan and R. Motwani, Coloring away communication in parallel query optimization, *in* “Proc. of the 21st Conference on Very Large Databases (VLDB’95),” 1995.

21. "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.

22. Intel Corporation "i960 Hx Microprocessor Developer's Manual," September 1998, Order Number: 272484-002, Intel, Santa Clara, CA.

23. Y. Ioannidis *et al*., Parametric query optimization, *in* "Proc. of the 18th Conference on Very Large Databases (VLDB'92)," August 1992.

24. W. de Jonge, M. F. Kasshoek, and W. C. Hsieh, The logical disk: A new approach to improving file systems, *in* "Proc. of the 14th ACM Symp. on Operating Sys. Principles (SOSP'93)," pp. 15–28, Dec. 1993.

25. K. Keeton, D. A. Patterson, and J. M. Hellerstein, The case for intelligent disks (IDISKS), *SIGMOD Record* **27**(3) (1998).

26. S. C. Lin, D. C. P. Smith, and J. M. Smith, The design of a rotating associative memory for relational database applications, *Trans. Database Systems* **2** (March 1976), 53–75.

27. D. E. E. Long, B. R. Montague, and L. Cabrera, Swift/RAID: A distributed RAID system, *Comput. Systems* **7**(3) (1994), 333–359.

28. 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.

29. 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,, Sept. 1999.

30. D. Murphy, "Building an Active Node on the Internet," Technical Report, MIT-LCS-TR-723, Master of Engineering thesis, MIT, Cambridge, MA, May 1997.

31. E. A. Ozkarahan, S. A. Schuster, and K. C. Smith, RAP—Associative processor for database management, *in* "Proc. AFIPS Conference," Vol. 44, pp. 379–388, 1975.

32. R. Ramakrishnan, "Database Management Systems," McGraw–Hill, New York, 1998.

33. 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.

34. 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.

35. C. Ruemmler and J. Wilkes, An introduction to disk drive modeling, *IEEE Comput*. **27** (March 1994), 17–28.

36. Siemens Microelectronics, Inc., "TriCore Architecture Overview Handbook," February 1999.

37. M. Stonebraker *et al*., A wide-area distributed database system, *VLDB J*. **5** (January 1996), 48–63.

38. D. L. Tennenhouse and D. J. Wetherall, Towards an active network architecture, *Comput. Comm. Rev*. **26** (April 1996).

39. 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.

40. P. Trancoso, J. L. Larriba-Pey, Z. Zhang, and J. Torrellas, The memory performance of DSS commercial workloads in shared-memory multi-processors, *in* "Proc. International Symposium on High Performance Computer Architecture (HPCA'97)," San Antonio, TX, Feb. 1–5.

41. Transaction Processing Performance Council, "TPC Benchmark D Standard Specification Revision 2.1," February 1998.

42. M. Uysal, A. Acharya, and J. Saltz, Evaluation of active disks for decision support databases, *in* "Proceedings of the 6th International Symposium on High-Performance Computer Architecture," Toulouse, France, January 10–12, 2000, to appear.

43. R. Van Meter, S. Hotz, and G. Finn, Derived virtual devices: A secure destributed file system mechanism, *in* "Proc. 5th NASA Conf. on Mass Storage System and Technologies," Sept. 1996.

44. R. Y. Wang, T. E. Anderson, and D. E. Patterson, Virtual log based file systems for a programmable disk, *in* "Proc. Third Symposium on Operating Systems Design and Implementation (OSDI'99)," February 1999.
45. R. Winter and K. Auerbach, Giants walk the earth: The 1997 VLDB Survey, *Database Program. Design* **10** (Sept. 1997).
46. R. Winter and K. Auerbach, The big time: The 1998 VLDB Survey, *Database Program. Design* **11** (Aug. 1998).
47. A. Yu and J. Chen, "The POSTGRES95 User Manual," Computer Science Div., Dept. of EECS, University of California at Berkeley, July 1995.