An Experimental Evaluation of Smart Disk Architectures Using DSS Commercial Workloads

Gokhan Memik, Mahmut T. Kandemir and Alok Choudhary

Techical Report No. CPDC-TR-9909-015

September 1999

Center for Parallel and Distributed Computing
Department of Electrical & Computer Engineering
Nortwestern University
Technological Institute
2145 Sheridan Road, Evanston, IL, 60208

An Experimental Evaluation of Smart Disk Architectures Using DSS Commercial Workloads

Gokhan Memik, Mahmut T. Kandemir and **Alok Choudhary**
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208
{memik, mtk, choudhar}@ ece.nwu.edu

Abstract

Smart disk systems with large storage capacities and growing computational power are becoming increasingly attractive. The idea is to perform parallel and filtering-type of data intensive computations on disks, close to data, thereby offloading the host processor and increasing the aggregate system power.

In this paper, we evaluate in detail the performance of a host-based architecture and a smart disk system using queries from Decision Support System (DSS) databases. We show how to optimize the execution of the whole query being implemented by bundling frequently occurring patterns together and posting the bundle to the smart disks in a single invocation that reduces unnecessary communication between host and smart disks. Using an accurate simulator that can simulate the host-based as well as smart-disk system, we illustrate that the smart disk architecture brings substantial benefits in terms of overall query execution times. In particular, in our base configuration, we are able to achieve improvements over the traditional architecture by as much as 83.5% using representative queries from the TPC-D benchmark suite.

1 Introduction

Many large-scale applications are data intensive in nature and require manipulation of huge data sets such as multi-dimensional scientific data, image files, satellite data, data base tables, and digital libraries. Apart from high computational requirements, these applications typically involve the transfer of large amounts of disk-resident data back and forth between the secondary storage devices and processing units. It is interesting to observe that in many of these applications the bulk of the computation on data can be performed where the associated data reside. The traditional way of processing, however, is to bring all the required data to where the computation will be performed.

Smart I/O devices, such as *smart disks*[1] with embedded processors, ASIC, and the capability to process and filter data close to where it resides have the potential to address this issue by reducing data movement, and thereby, reducing bandwidth requirements. This is based on the premise that in the near future 100 GBytes disks with embedded processors with hundreds of MIPS and a few hundred Mbytes of RAM will be available [13]. Therefore, it is attractive to consider solutions where computations can be moved closer to data to take advantage of these features. However, such approaches introduce many new challenges and problems that must be addressed. Previous work on this area addresses the architecture design and operating system design related issues [27,32,17]. Acharya et al. [1], on the other hand, focus on the implementation of individual database operations such as selection and join operators and of some scientific kernels on smart disk architectures. However, in order to gain further insight to the capabilities of such architectures, we need to evaluate them extensively using benchmarks from different software domains. The database applications lend themselves very well to smart disk architectures as many database operations are inherently filtering type. As a simple example, a selection operation (without an index) accesses (scans) a number of database tuples and *filters out* the ones that do not satisfy a selection criterion. Consequently, instead of performing the selection operation on the host processor (and transferring all the tuples across the I/O interconnect before doing so), we can perform it on disk using a cheap embedded processor. However, in general, database queries also involve selection operations with indexes. In that case it may not be particularly useful to utilize a smart disk architecture as in general the disk accesses in an indexed selection operation is random and highly selective. Also, database queries may involve multiple operations. It is not clear how the techniques developed for optimizing individual database operations [1, 2] can be integrated within a whole query optimization. Therefore, in order to judge the usefulness of smart disk architectures objectively we consider complete queries, which involve a large spectrum of database operations.

In this paper we analyze in detail the performance of several queries representing Decision Support System (DSS) databases. Specifically, we evaluate six queries from the TPC-D benchmark suite [30] to compare a smart disk architecture with a more traditional host processor based system. The queries that we use cover different combinations of select, join, sort, group-by, and aggregate

[1] We use the term *smart disks* to refer to a class of architectures that put substantial computational power on disks, such as [1], [17] and [27].

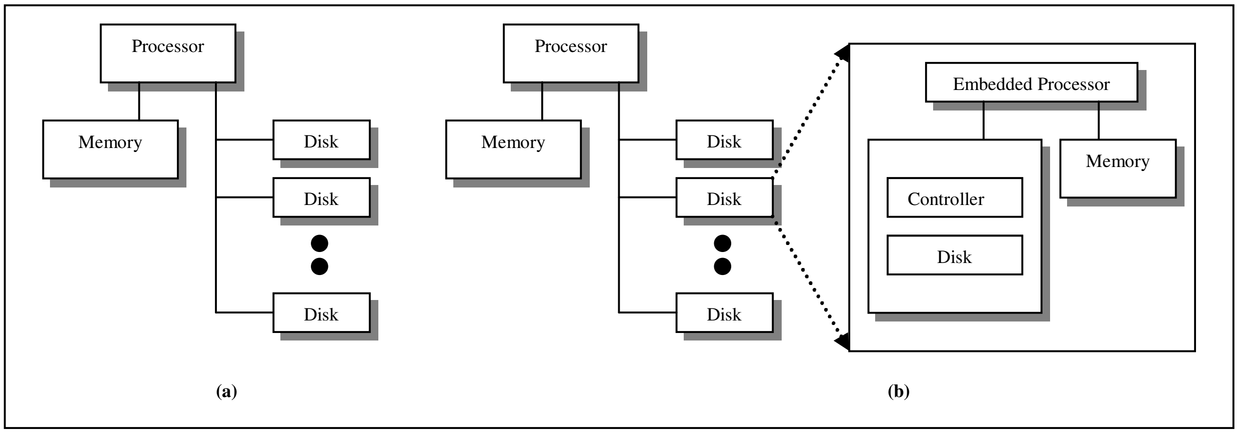


Figure 1: (a) Traditional architecture. (b) Smart disk architecture.

operations, thereby, enabling us to measure the potential of the smart disk architecture on DSS workloads. Our results reveal that smart disk architectures have the potential to deliver high levels of performance for different DSS queries under different experimental parameters such as processor speed, available memory size, and number of disks. In particular when we fix the processing power (in terms of MHz) at a specific value for both the smart disk architecture and the traditional host-based system, the former outperforms the latter by as much as 59%. Data in very large-scale applications can range in hundreds of GBytes per day and required GFLOPS range of processing [16]. In addition to demanding significant processing power, these applications require fast I/O and significant memory and I/O bandwidth. For example contemporary DSS databases process up to 4.5 TBytes of data, consisting of up to 50 billions of rows [33, 34]. These challenges require innovative approaches in architecture, software, and algorithm areas. Traditional approaches, which rely on technology dependent increase in bandwidth, capacity and processing power may not be sufficient to solve these problems. Considering our performance results presented in this paper, the employment of smart disk architectures in systems processing the DSS workloads seems attractive.

In the following section we describe the smart disk architecture and compare it with a traditional host-based system. In Section 3 we introduce the selected TPC-D queries and explain how we can execute them on the smart disk architecture. We also explain how individual database operations can be implemented on our platform. In Section 4 we present our simulator and show using validation that it is highly accurate. In Section 5 we describe our experimental platform, define the methodology used in the experiments, and present the simulation results. In Section 6 we discuss related work and in Section 7 we conclude the paper with a summary and an outline of the on-going research.

2 Smart Disk Architecture

Smart disk architecture contains a host processor and a smart disk sub-system. The latter consists of an embedded processor, a controller (depending on the architecture), a disk, and some amount of DRAM (see Figure 1(b)). The programming model for such an architecture is to split the applications' computations and data in such a way that the parts that are I/O-intensive and result in considerable reduction in data (we call these *filtering* operations because in a way most of these operations filter data and the resulting data is much smaller) are performed on disk-processors (and in parallel when data in multiple disks are involved), while compute-intensive and coordination tasks are performed on the host-processor. In other words, a smart disk system exploits the increasing computational power on disk devices by allowing the filtering type of functions to operate directly on disks, close to the data, and returning only the relevant fraction of data to the host. This down-loading application code and executing it on disk devices does not only reduce the network traffic but also offloads the host processor and increases the aggregate system power.

For scientific codes, the language and programming model for this architecture is still in its infancy. The important issues to be addressed are the language constructs that allow easy data partitioning across smart disks and the host, code safety, code loading to the smart disks, and the interaction between the operating system and the disk-resident code [26]. Acharya et al. [1] propose a stream-based programming model in which the smart disk resident code and the host resident code interact using streams.

For database applications, however, there is some previous research to build upon. First, there is a vast amount of literature on database machine architectures which were studied long time back [6, 24, 5, 21]. Compared to these architectures, the smart-disk system does not use any special purpose hardware and relies mostly on COTS components. Special-purpose components with their modest performance and high costs eventually led to the demise of earlier database machines. Today's interconnection networks are also very fast as compared to those of 20-25 years ago. Also, we take efficient query optimization techniques into account. Unlike most of the old database machines we do not necessarily perform all relational operations on disks; rather we can take selectivity and cost of the database operations into account and try to come up with efficient query optimization techniques which divide the query into host-resident and smart-disk-resident parts. Second, in the last decade parallel execution of database queries made significant progress [9]. The relational database operators are amenable to both *pipeline parallelism,* in which the computation of one operator proceeds in parallel with another, and *partitioned parallelism,* in which operators are replicated in each processing unit. Considering each smart disk (with its embedded processor) as a processing unit in the parallel database sense, we can expect at least some query parallelization techniques to be adapted to the smart disk architecture.

Finally, we remark on some architectural specifications. Compare the smart disk architecture shown in Figure 1(b) with the traditional host-based architecture given in Figure 1(a). In todays' standards the host CPU in Figure 1(a) is between 200-600 MHz with up to 1 GB of DRAM. The I/O interconnect is between 150-300 MB/s. The embedded processor in Figure 1(b), on the other hand, is 100 to 300 MHz. Keeton et al. [17] estimate that even including the extra cost of a fast serial link that connects smart disks together, the total cost of a smart disk architecture will be on the order of 10% of total cost the cost of a traditional host based system.

3 DSS Queries

We use *six* queries from the standard TPC-D benchmark [30]. This benchmark suite contains 17 read-only queries and 2 update queries, most of which are large and complex. Table 1 shows the read-only queries we selected and the operations performed by them. An ‘X’ denotes that the query in question involves the relevant operation. For example, the query Q_{12} involves sequential scan (SS), indexed scan (IS), merge join (M), sort, group-by and aggregate operators. Our choice of these queries is motivated by our desire to cover each operation at least once.

The benchmark also specifies the possible values for the parameters in the query. Thus, the possibility of a tuple being selected by the query is fixed. Different queries have different selectivities. For example Q_{13} selects all the tuples from one of its input tables, whereas Q_{12} selects one in 200 tuples from a table called `lineitem`. Our choice of the queries ensures that both low selectivity and high selectivity queries are evaluated. Another property of the selected queries is that there are low and high selectivity queries for both indexed and sequential scans.

In the following we first explain how each individual database operation (e.g., SS, sort) is executed on the smart disk platform, and then we discuss how a given query that involves a set of these database operations is executed as a whole.

Table 1: The read-only TPC-D queries that we used and their operations. The operations are sequential scan (SS), indexed scan (IS), nested loop join (NL), merge join (M), hash join (H), sort, group (Gr.), and aggregate (Agg.).

Query	Select		Join			Sort	Gr.	Agg.
	SS	IS	NL	M	H			
Q1	X					X	X	X
Q3		X	X			X	X	X
Q6	X							X
Q12	X	X		X		X	X	X
Q13	X	X	X			X	X	X
Q16		X			X	X	X	X

3.1 Individual Database Operators

The implementations of SS, group-by, and aggregate are similar to those proposed by Acharya et al. [2]. For SS and aggregate, each smart disk performs the scanning (filtering) and aggregation locally and then transfers the results to the host processor. The host, in turn, combines the in-coming tuples (for SS) and aggregates (for aggregation) and reports the result. In the IS operation, each smart disk builds its index for the portion of the data that it owns. Using these indices the selected tuples are forwarded to the host and the host combines these results. The group-by operation computes a list of aggregates indexed by a set of attributes (fields). Hashing-based, sorting-based, and nested-loop-based algorithms exist. We use a hashing-based algorithm [12], in which for the first phase the local group-bys are performed by the smart disks. They then send the partial results to the host where these results are accumulated in the second phase.

The nested-loop-join operation (NL) is implemented in two phases as follows. In the first phase, each smart disk communicates one of its sub-tables to all the others. Thus, one of the tables is practically replicated in each smart disk. The table to be replicated is determined by the host machine. Then using the indices, each smart disk performs local nested joins and the qualifying tuples are sent to the host. The merge-join operation (M) is implemented very similarly. Instead of using index-based searches as in NL, the smart disks perform local merge operations. In the hash-based-join operation (H), first the local hashes are built in each smart disk. Then these hashes are sent to other smart disks to form a global hash table for one of the relations.

Table 1: Execution protocol for a smart-disk based system.

TASK	HOST MACHINE	SMART-DISK
Query Parsing	-Parse query	
Query Optimization	-Perform query optimization	
Query Fragmentation	-Fragment query into bundles	
Execution of the intermediate bundles	-Send the bundle to smart disks -Wait for the completion signal	-Receive the bundle -Execute the bundle -Store the result -Send the completion signal to host
Execution of the last bundle	-Send the bundle to smart disks -Wait for the results	-Receive the bundle -Execute the bundle -Send results to host
Combining the Results	-Combine the results of each smart disk	

Consequently, the local hash joins are executed in each smart disk and the constructed tuples are forwarded to the host. For the sort operation, we used an external local sort in each disk and the sorted results are transferred to the host where they are merged. Note that, for this operation, more sophisticated sort algorithms, such as NOW-sort [3] could also be adopted.

3.2 Whole Query Execution

3.2.1 Operator Bundling

In this section we explain how a whole query is executed on a smart disk system. The core of our approach is to *bundle* (where appropriate) a number of database operators and execute this bundle as a single operation on the smart disks. On one hand, the number of operations in a bundle should be maximized in order to minimize the message traffic between the smart disks and host. On the other hand, we would like to keep the number of operations in a bundle small so that, in case of an error, the host can unambiguously detect what parts of the query have been successfully completed so far. Although consecutive database operations can be bundled in a number of ways, currently we bundle only the operators occurring in a frequent pattern. In our bundles, each operator uses only the results generated by a previous operator in the bundle without the need of extensive communication. For example, two select operators whose results are joined immediately by a join operator (as in the query Q_{12}) are bundled with the join operation. The select operator can be SS or IS and the join operator can be NL, M, or H. As a second example, we bundle a group-by operation with a following aggregate operation (as in the query Q_3).

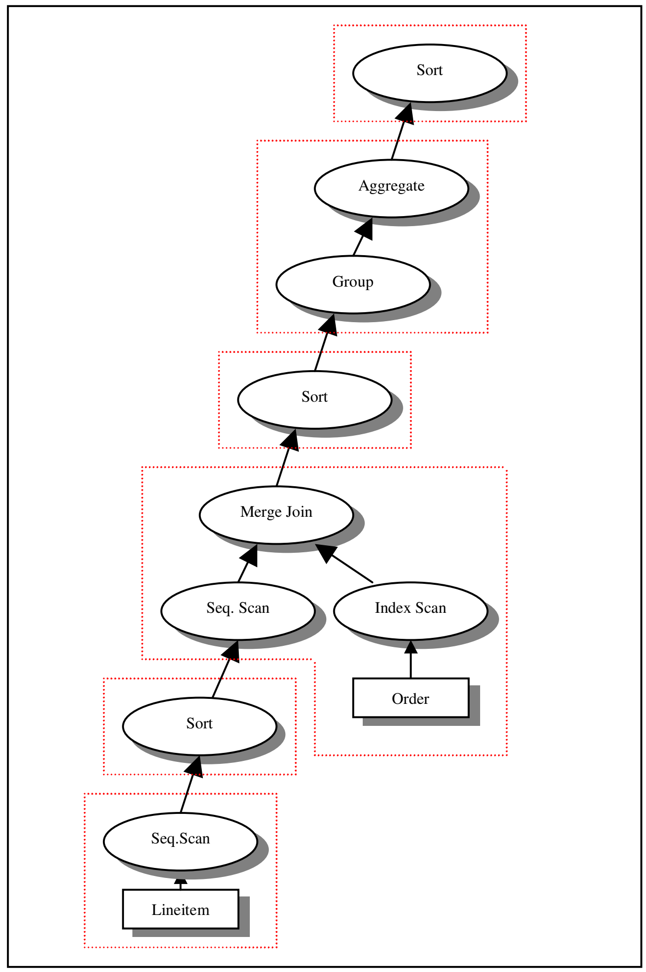


Figure 2: Query execution plan for Q_{12} (Each dashed box contains a bundle).

Query execution starts on the host machine, where the query is parsed and optimized. These steps produce a query plan tree [25]. Then the plan tree is fragmented by the host machine into bundles. That is, the host machine decides which operations in the plan tree are to be bundled together. Then, the host machine 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 to the amount of memory available, the results are stored either in memory or on disk. In the last bundle, the host machine instructs the smart disks to send the result to the host machine. The host machine receives these results and combines them. This generic protocol between the host and the smart disks is summarized in Table 2.

3.2.2 Example

Let us concentrate on the query Q_{12} whose SQL code is given in Figure 3 and whose query execution plan is shown in Figure 2 (this plan tree is from [29]). The query execution is started by the host machine and the query is divided into bundles (shown by the dashed boxes in Figure 2). A message is sent by the host telling the smart disks 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. The host also indicates that the results should be stored on the smart disks. Then, the table `lineitem` is sequentially scanned by the smart disks. Depending on the size of the resulting tuples, the resulting tuples in each smart disk can be cached in memory or may be written on the disk. After the selection operation is finished, a completion signal is sent to the host which also includes information about the size and the identification of the resulting table. Then, the next bundle, which contains the sort operation, is sent. In the message it is indicated that the results should be stored on the smart disks. When this bundle is done, the next bundle which contains a join and two select operations is forwarded to the disks. 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 made by the host machine and indicated in the bundle sent. The most important parameter in this decision is the size of the table. For this query, it is more beneficial to select the filtered `lineitem` table. The smart disks communicate with each other to replicate the filtered table. Then, the filtered `lineitem` table is sequentially traversed in each smart disk and for each element in this table, an element from the `order` table with the same `orderkey` is fetched and stored in an intermediate table. Again the resulting table is stored on the smart disks. After this bundle, the host sends an instruction for sorting the intermediate table. Following this step, the smart disks are instructed by the host to perform the group-bys and aggregates as a single bundle. We should mention that the an aggregation operation after group-by is also a commonly encountered pattern sequence, therefore these operations are also performed as a single bundle. In the last step, the host asks the disks to sort the intermediate results. But this time, it indicates that the results should be sent to the host machine. The partial results from the smart disks are collected by the host and combined to form the result of the entire query.

```
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(`[MODE1]`, `[MODE2]`)
  and l-commitdate < l-receiptdate
  and l-shipdate < l-commitdate
  and l-receiptdate >=`[DATE]`
  and l-receiptdate < `[DATE]` + `1' year
group by
  l-shipmode
order by
  l-shipmode;
```

Figure 3: The SQL code for the query Q_{12} from TPC-D.

4 Simulator Validation

To conduct the experiments, we have developed `DBsim`, an execution driven simulator, which simulates the database operations for both of the architectures. `DBsim` has detailed models for disks, I/O interconnects and processors.

To simulate the disk behavior, `DBsim` uses `DiskSim` developed by Ganger et al [19]. `DiskSim` is an efficient and accurate disk system simulator. It includes modules for simulating disks, intermediate controllers, buses, device drivers, and request schedulers. The *demerit figure*, as defined in [20], of `DiskSim` in worst-case is only 2.0% of the corresponding average response time.

`DBsim` is capable of simulating the sequential and indexed scan, nested loop, merge, and hash join, sort, group, and aggregate operations. Besides the architectural parameters, it requires database specific parameters like total data size, location of data, index properties, tuple size, size of the resultant table and the size of the resultant tuples. The *host simulator* is a program which reads the corresponding parameters from a configuration file and calls `DBsim` with the appropriate values. Note that the memory and processor properties are read by the host simulator and are passed to `DBsim`. It also checks the remaining memory size after each step of the query plan tree and passes this value to `DBsim`. The *smart-disk simulator* is a parallel program, which similarly reads the parameters from a configuration

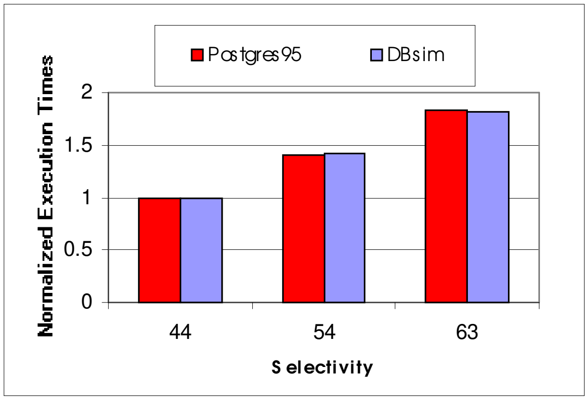


Figure 4: Relative execution times for Q_3, s = 0.1

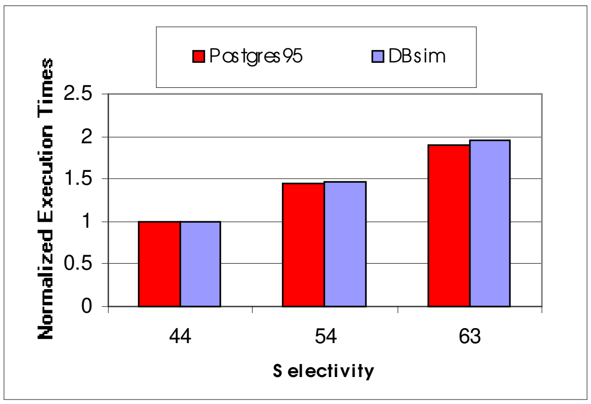


Figure 5: Relative execution times for Q_3, s = 0.05

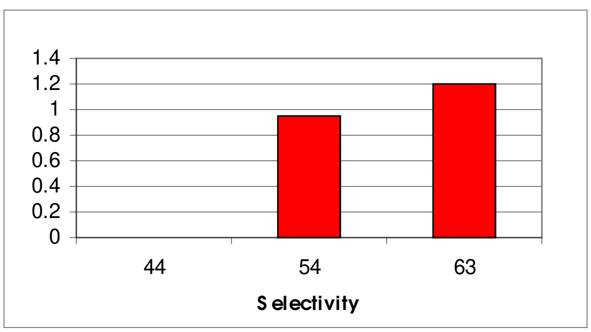


Figure 6: Absolute error for changing the selectivity in Q_3.

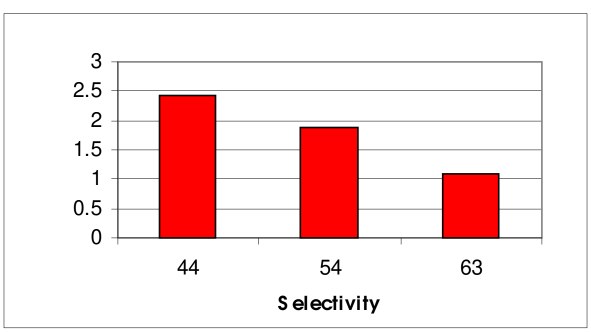


Figure 7: Absolute error for changing the database size in Q_3.

file. Each processor calls `DBsim` with the corresponding parameters. Then, according to the results obtained from `DBsim`, the processors communicate with each other using message passing.

To validate the `DBsim` simulator we conducted several experiments. We compared the response times obtained from `DBsim` against the values we obtained by executing the queries on Postgres95 database system [18]. Postgres95 is installed on a IBM RS/6000 workstation with three IBMRISC DFHSS4W 4.5GB 16 Bit SCSI disk drive. We measured the response times for the queries Q_3 and Q_6 for two different data sizes and three different selectivities. We ran experiments for scale factors of s = 0.1 and s = 0.05[2]. The selectivities ranged from 2% to 4% and from 44% to 63% for Q_6 and Q_3, respectively. These values are found by substituting different values for the required parameters defined in [30].

Figure 4 gives the relative response times of `DBsim` and Postgres95 for Q_3 with the scale factor s=0.1. The figure reveals that the behavior of Postgres95 is closely matched. Figure 5 gives the response times for s=0.05. Figure 6 presents the percentage errors (%) observed for changing selectivity with the same database size. The values are found by dividing the difference of increase rates to the increase rate of Postgres95. The results were similar for the scale factor of s=0.05. Finally, Figure 7 gives the percentage errors for changing the database size. These values are found similarly. The results show that the worst-case error of `DBsim` for Q_3 is 2.4%. The average error of `DBsim` in Q_3 for different selectivities and database sizes is 1.4%. The average error for Q_6 was 1.1%. The experiments with several architectural parameters gave similar results. Overall, we can conclude that the simulator is highly accurate.

5 Experiments

In this section we present the simulation results. We first give the results for a base configuration with typical architectural parameters. Subsequently, we vary different parameters and observe their impact on the performance. Finally, we present some results to illustrate the performance when the total power of the system is fixed at a specific value for both the architectures.

5.1 Base Configuration

Our base configuration is given in Table 3. In each of the results (Figures 8 through 22) presented later in this section we compare two different architectures: *a traditional architecture with a conventional disk subsystem and a smart disk architecture*. As far as the base configuration is

[2] *Scale factor* corresponds to the total database size. For instance, s = k means that the total size of the tables in the TPC-D database is k GB.

concerned, the conventional system has a 500 MHz host processor and eight 5 GB disks. The smart disk system on the other hand, consists of 8 smart disks, each with 200 MHz embedded processor and 5 GB of disk. It is assumed that the smart disks have fully connected TB3 switches between them with a 150 MB/sec peak bandwidth.

Figure 8 presents the *normalized execution times* for the six queries for both the architectures. In this and the following figures, x-axis denotes the queries and y-axis denotes the execution times normalized with respect to those of the host-based system. ***The left bar of each query in the graph shows the normalized time for the host-based architecture whereas the right bar denotes that for 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 smart disk system is divided into three parts, the computation time, the I/O time, and the communication time. The computation time denotes the time spent by all processors (host and smart) in executing the query code. The I/O time, on the other hand, is the time spent in I/O by host (including the data transfer time from disks in the host-based system) and smart disks (in the smart disk system). Finally, the communi-cation time is the time spent in communication between smart disks and host and between smart disks themselves. The results show that in the base configuration the smart disk system brings an improvement (reduction in execution time) between 55.4% and 83.5%, averaging 71.4%; i.e., the response time decreases by 71.4% in average with respect to the host-based architecture. We also observe that the time spent for communication is very small as compared to I/O and computation times due to the small size of the results delivered by the smart disks to the hosts, which was our aim by designing our query execution protocol. It is important to notice that the smart disk system reduces I/O as well as the computation time due to exploiting inherent parallelism in queries.

5.2 Varying Simulation Parameters

We vary different parameters to investigate their effect on the performance. Table 4 shows different variations (v1 through v9) that we experiment with. In each case all the parameters are the same as in the base configuration *except* the relevant parameter. For instance, in v1, only the host processor speed is increased, the remaining parameters remain the same as given in Table 3. It should be noted from Table 4 that we experiment with three variations for relation size and selectivity (`small`, `medium`, and `large`, including the base configuration). The interpretations of these parameters are given in Table 5.

5.2.1 Varying Host-related Parameters

We see from Figure 9 that when we use a faster processor on the host side, we have a 61% improvement over the traditional architecture on average. Increasing the memory size on the host side (Figure 10), instead, results in 68% improvement. Comparing these last two results with that of the base configuration, we see that a better host processor makes a slight improvement on the smart disk architecture whereas the improvement on the traditional system is considerable. Figure 11, on the other hand, shows the impacts of a faster I/O interconnect. The average improvement for this case is 70.9%.

5.2.2 Varying Disk-related Parameters

We now focus on the disk-related parameters. Figure 12 shows how a faster embedded core affects the performance. As compared to the base case (Figure 8) the effectiveness of the smart disk system is increased by 30%, resulting in 79.4% improvement compared to the traditional architecture, on the average. This can be attributed to the fact that most of the work in a smart disk system is done by the smart disks. Increasing the memory size of the smart disks has a similar impact as illustrated in Figure 13. This is due to the fact that a larger memory allows a smart disk to keep more intermediate data on-line (instead of writing and possibly re-reading from its local disk). The highest increase in performance is seen in Q_3, where the join operation requires considerable amount of data to be processed.

The next two experiments investigate the sensitivity with respect to page size (Figures 14 and 15) and number of disks (Figures 16 and 17). The average improvements of smart disk architecture are 70.9% and 75.0%, for 4-KB page size and 16-KB page, respectively. As the page size increased, more redundant data will be transferred to the processing element. This puts an excessive demand on the disk subsystem, resulting in higher improvements for the smart disk architecture. With 4 and 16 disks, the smart disk

Table 3: Base configuration for the experiments.

Parameter	Value
Host CPU	500 MHz
Host Memory	258 MB
I/O Interconnect	200 MB/s
Disk CPU (including host)	200 MHz
Disk Memory	32 MB
Smart Disk Interconnect	150 MB/s
Data Page Size	8 KB
Number of Disks	8
Rotation Speed	5400 rpms
Min. Seek Time	1.62 msec
Mean Seek Time	8.46 msec
Max. Seek Time	21.77 msec
Relation Size	`medium`
Selectivity	`medium`

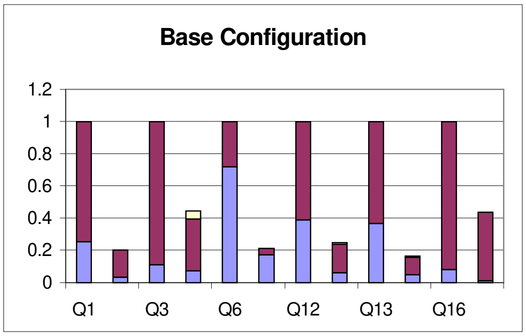


Figure 8: Relative execution times for the base configuration.

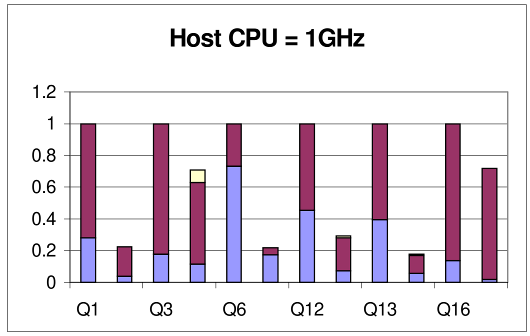


Figure 9: Relative execution times for faster host processor.

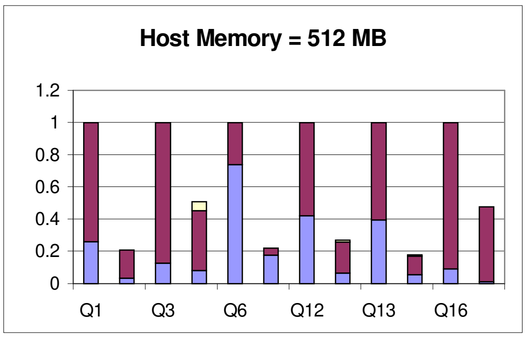


Figure 10: Relative execution times for larger host memory size.

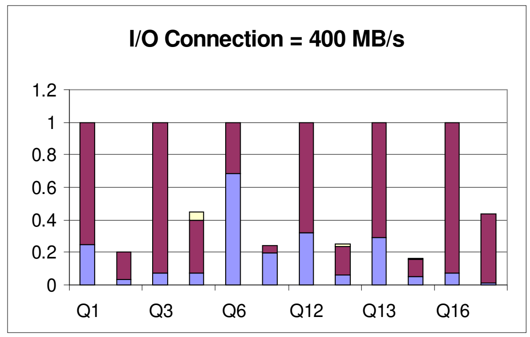


Figure 11: Relative execution times for faster I/O interconnection.

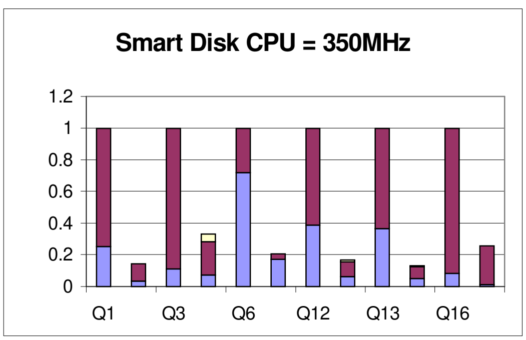


Figure 12: Relative execution times for faster smart disk processor.

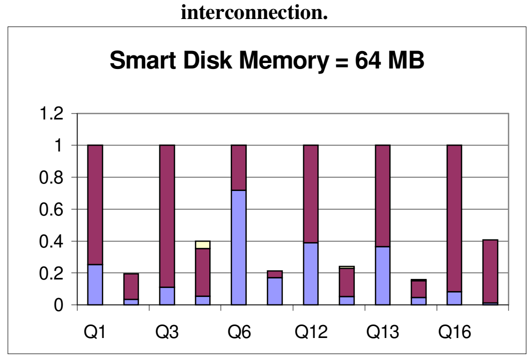


Figure 13: Relative execution times for larger size smart disk memory.

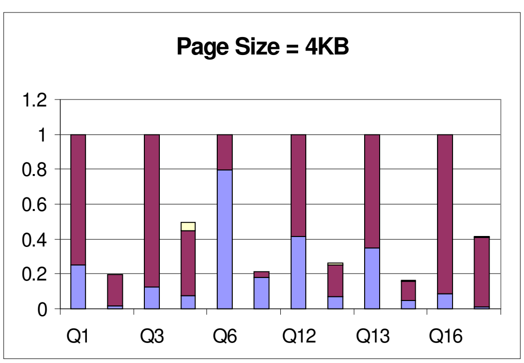


Figure 14: Relative execution times for page size of 4 KB.

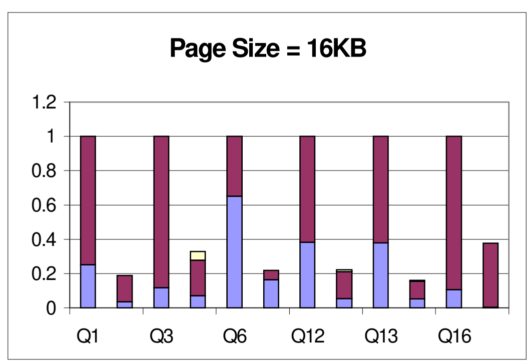
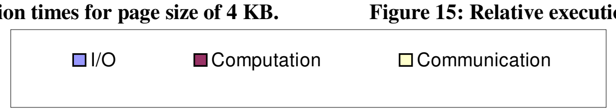


Figure 15: Relative execution times for page size of 16 KB.



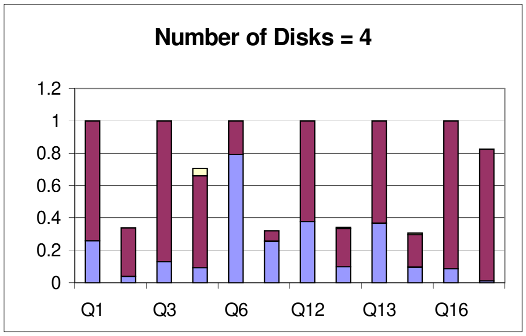


Figure 16: Relative execution times for four disks.

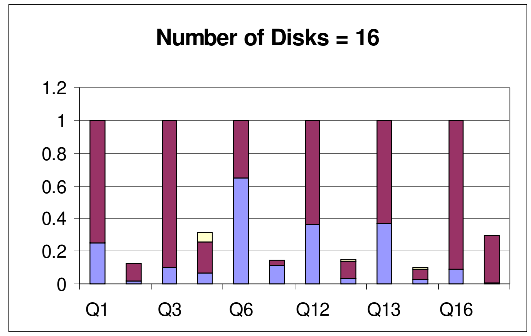


Figure 17: Relative execution times for 16 disks.

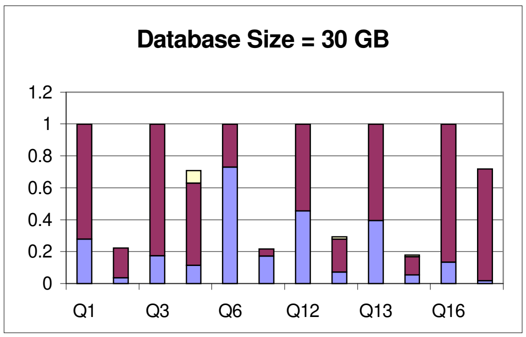


Figure 18: Relative execution times for `large` database size.

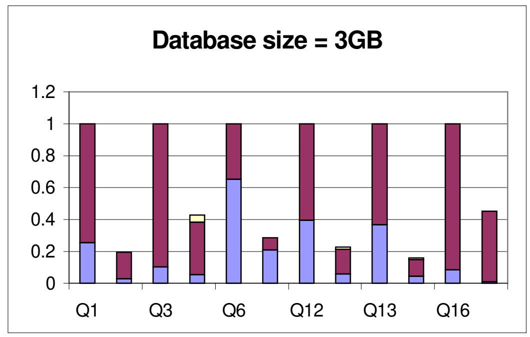


Figure 19: Relative execution times for `small` database size.

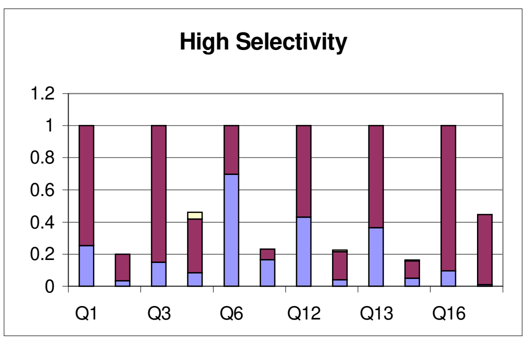


Figure 20: Relative execution times for `large` selectivity.

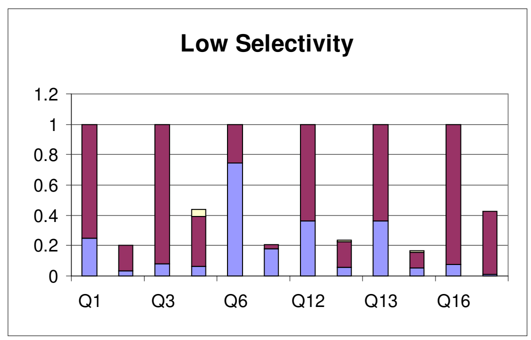


Figure 21: Relative execution times for `small` selectivity.

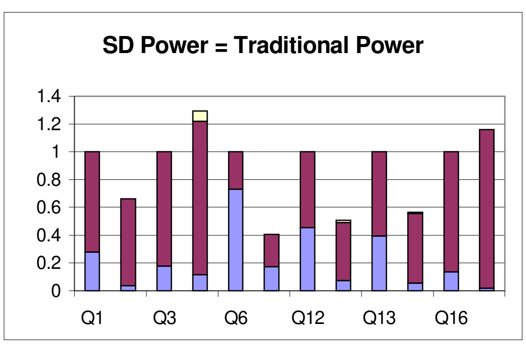


Figure 22: Relative execution times for balanced computational power.

- In each figure, x-axis denotes the queries and y-axis denotes the normalized execution times.
- For each query, the left bar shows the execution time of the host-based system whereas the right bar gives that of the smart disk system.
- For the host-based system, the I/O portion includes the data transfer time between the host and disks.

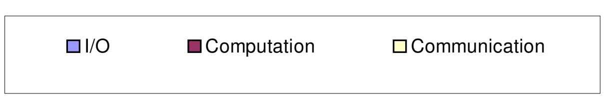


Table 4: Variations in simulation parameters with respect to the base configuration given in Table 3.

VARIATIONS ->	v1	v2	v3	v4	v5	v6	v7	v8	v9
Parameter	Host CPU Speed	Host Mem Size	I/O Conn. Speed	Disk CPU Speed	Disk Mem Size	Data Page Size	# of Disks	Relation Size	Selectivity
New Value	1 GHz	512 MB	400 MB/s	350 MHz	64 MB	4K, 16 K	4, 16	`small,large`	`small,large`

Table 5: Relation sizes and selectivities.

Number of tuples in the tables (M denotes *millions* K denotes *thousands*)

	Q_1	**Q_3**	**Q_6**	**Q_{12}**	**Q_{13}**	**Q_{16}**
`small`	18M	18M x 4.5M x 0.45M	18M	18M x 4.5M	4.5M x 0.45M	2.4M x 0.6M x 30K
`medium`	60M	60M x 15M x 1.5M	60M	60M x 45M	15M x1.5M	8M x 2M x 0.1M
`large`	180M	180M x 45M x 4.5M	180M	180M x 45M	45M x 45M	24M x 6M x 0.3M

Total size of the tables

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

Selectivities (in percentages)

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

architecture outperforms the traditional architecture by 52.7% and 81.2%, respectively. For smart disk architecture, doubling the number of disks results in an improvement of a factor of 1.82 in average. On the other hand, the improvement factor for the traditional architecture is 1.12. These results show good scalability for the smart disk architecture. Acharya et al. [1] report that some individual operations (e.g., sort operation) for large queries do not scale well. In our experiments this is not observed, because operations that scale well dominate the whole query, making the system scale well (The minimum scalability observed is in Q_3, for which the average scalability is 1.68).

5.2.3 Varying Database Parameters

The last set of variations are in database parameters. When we increase the relation size we have better improvements (Figure 18). For example, with `large` size we have 74.9% improvement with smart disks. This can be expected as in this case we have more data accesses to optimize. As expected, increasing selectivity (Figure 20) reduces the effectiveness as it causes more communication. One of the advantages of the smart disk architecture is that the irrelevant data (e.g., database tuples that will not appear in the result) will not be transferred to the processing unit. When we increase the selectivity, the proportion of the unselected data decreases. This reduces the advantage of smart disk architecture. This behavior of selectivity we have observed is similar to the results obtained by Riedel et al. [26].

5.3 Balancing the Computational Power

In this part we compare the performance of a single host machine with disks against *less powerful* host machine with various configurations of smart disks attached. We used a host based system with a 1 GHz processor; the other parameters are as in Table 3. The smart disk system, in comparison, has a 200 MHz host processor and 8 smart disks each with 100 MHz embedded processor. In other words, we fixed the total computational power in both the architectures at 1 GHz. The results given in Figure 22 reveal that smart disk system achieves improvements even under this less favorable condition. For example, we observed an improvement of 59% in Q_6. Even though the average improvement was 23% over the traditional architecture, for two out of six queries, the smart disk architecture took longer time to execute; that is, the gains obtained from parallel execution on smart disks got outweighed by not utilizing the host in the smart disk system fully and by the extra communication overhead.

6 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. [6] proposed putting a processor per disk head. The others offered processor per track and processor per disk [24, 5, 21]. Unfortunately, the special-purpose components eventually led to the demise of earlier database architectures. An important difference between these works and ours is that we take efficient query optimization techniques as well as the experience in parallel query optimization into account. We also consider selectivity and cost of the database operations and try to come up with efficient query optimization techniques for smart disks.

Previous research demonstrated how to exploit a smart disk architecture which integrates significant processing power and memory into a disk drive and down-loads application specific code to disks [1, 2, 17, 26, 27]. Acharya et al. [1, 2] hand-optimized a number of isolated database operators to run on the *Active Disk* architecture. Their simulation results show that the processing power on disks might be useful for data intensive, filtering type of applications. Riedel et al. [27] also focused on the active disk architecture and discussed several types of applications (from data mining and multimedia) that would benefit from the capability of processing data on disks. In [26] they also presented performance numbers for selection and parallel sort operations. Keeton et al. [17] present a disk architecture called *Intelligent Disks* (IDISKs) that puts significant processing power at the disks to overcome the I/O bus bottleneck of conventional systems. The main idea is to off-load computation from expensive host processors. As compared to the active disks [1], the IDISKs are meant to be more general purpose. Our work builds upon and extends the above work and is different in the following aspects.

- Unlike [17], we present an evaluation of smart disks using representative queries from the DSS databases. Keeton et al. [17] focus more on a vision of processing using intelligent disks.
- Unlike [1, 2] and [26, 27] we focus on complete database queries and develop a query optimization technique that takes interactions among individual database operators into account. For this purpose, we introduce *operator bundling* in which a number of relevant database operators are bundled and sent by the host to the smart disks in a single function call. Deciding which operations are to be bundled is based on the cost of the operations and various architectural parameters. The bundling of operations requires a more extensive programming model, because in earlier models the result of each operation needs to be returned to the host. In the bundling model, on the other hand, the result of one operation in a bundle can be reused directly on smart disk by a subsequent operation in the same bundle without the need of host interference.
- We also report numerical results for performance of *complete queries* from the TPC-D benchmark [30] and compare the smart disk architecture with a traditional host-based system.

Derived Virtual Devices [31] 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 host 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 [11].

Virtual disks [4] and logical disks [8] 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 [10, 22, 15] study the impact of device-level and server-level parallelism on I/O performance. These systems still treat disks as merely as storage. Our work will extend the function of any disk on the network to encompass local processing at the disk.

Putting intelligent logic into a single chip has also been applied to memories. For example, Berkeley's IRAM (intelligent RAM) project [36] implements such an integration which reduces memory latency and increases memory bandwidth.

Active Networks [28, 23] envisions code migration to the network to improve QOS. These projects [7, 14] 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.

7 Conclusions

A growing trend in computer architecture is to put increasing computational power to the embedded systems [26, 35]. A promising use of this excess power on disk subsystems is to offload user-defined code to the disks and perform the computation directly on drives, close to where data reside. This can significantly reduce the traffic on the I/O network and enable more effective use of the host processor. We evaluated through extensive simulations a traditional host-based system and a smart disk architecture using representative queries from Decision Support System (DSS) databases. Our experiments with different processor speeds, memory sizes, and database parameters reveal that the smart disk architecture can bring impressive speedups for typical DSS queries. The work-in-progress includes design and implementation of automatic query optimizers that can handle other types of queries as well (e.g., update queries) and investigation of different architectures that involve multiple hosts and smart disk systems.

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

[4] 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 no. RC 19843,* Nov. 1994.

[5] E. Babb. Implementing a relational database by means of specialized hardware. *ACM Transactions on Database Systems*, Vol. 4, No 1., March 1979.

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

[7] S. Bhattacharjee, K. Calvert, and E. W. Zegura. Implementation of an active networking architecture. White paper presented at *Gigabit Switch Technology Workshop*, Washington University, St. Louis, July 1996.

[8] W. de Jonge, M. F. Kasshoek, and W. C. Hsieh. The logical disk: a new approach to improving file systems. In *Proc. of 14th ACM Symp. on Operating Sys. Principles (SOSP'93),* pp. 15-28, Dec 1993.

[9] 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.

[10] A. Drapeau et al. RAID-II: A high-bandwidth network file server. In *Proc. of 21st Ann. Symp. on Computer Architecture (ISCA'94),* pages 234-244, April 1994.

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

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

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

[14] 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.

[15] 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),* pages 29-43, Dec. 1993.

[16] 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.

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

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

[19] 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.

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

[21] S. C. Lin, D. C. P. Smith, and J. M. Smith. The design of a rotating associative memory for relational database applications. *Transactions on Database Systems*, Vol 1. No.1, March 1976, pp. 53-75.

[22] D. E. Long, B. R. Montague, and L. Cabrera. Swift/RAID: A distributed RAID system. *Computing Systems*, 7(3):333-359, 1994.

[23] D. Murphy. Building an active node on the internet. *Technical Report MIT-LCS-TR-723,* Master of Engineering Thesis, MIT, Cambridge, MA, May 1997.

[24] A. Ozkarahan, S. A. Schuster, and K. C. Smith. RAP - Associative processor for database management. In *Proc. AFIPS Conference*, Vol 44, 1975, pp. 379-388.

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

[26] 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.

[27] 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.-TR-358-98, Dept of Electrical Engineering and Computer Science, Feb 1998.

[28] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *Computer Communication Review*, Vol. 26, No. 2, April 1996.

[29] P. Trancoso, J.-L. Larriba-Pey, Z. Zhang, and J. Torrellas. The memory performance of DSS commercial workloads in shared-memory multi-processors. In *the Proc. InternationalSymposium on High Performance Computer Architecture (HPCA'97),* San Antonio, TX, Feb 1-5.

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

[31] 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.

[32] R. Y. Wang, T. E. Anderson, 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.

[33] R. Winter and K. Auerbach. Giants walk the earth: the 1997 VLDB Survey, *Database Programming and Design* Vol 10, No 9, Sept 1997.

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

[35] S. Heath. *Embedded System Design,* Butterworth-Heinemann, Jordan Hill, Oxford, UK, 1997.

[36] D. Patterson et al. A case for Intelligent RAM, *IEEE Micro*, Vol 17, No. 2, March-April 1997. pp. 34-44