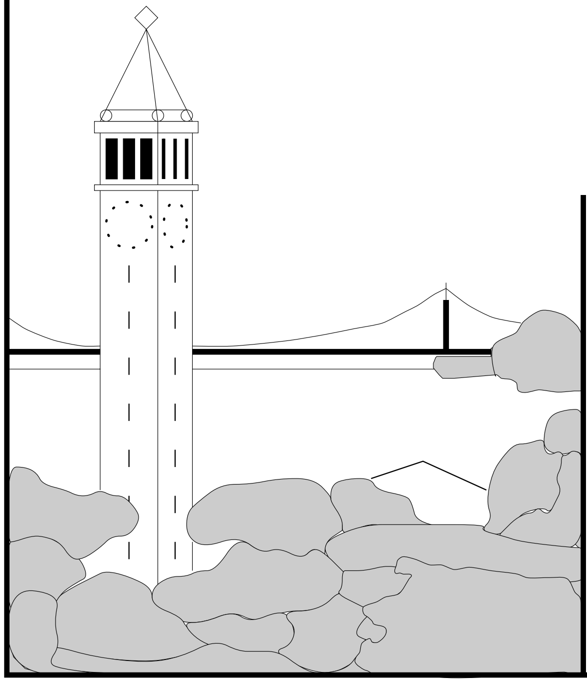
Projecting the Performance of Decision Support Workloads on Systems with Smart Storage (SmartSTOR)

Windsor W. Hsu[⋆][†]
Alan J. Smith[†]
Honesty C. Young[⋆]

[⋆]*IBM Research Division*
IBM Almaden Research Center
San Jose, CA 95120
{*windsor,young*}*@almaden.ibm.com*

[†]*Computer Science Division*
University of California
Berkeley, CA 94720
{*windsorh,smith*}*@cs.berkeley.edu*



Report No. UCB/CSD-99-1057

August 1999

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Projecting the Performance of Decision Support Workloads on Systems with Smart Storage (SmartSTOR)

Windsor W. Hsu[⋆†]
Alan J. Smith[†]
Honesty C. Young[⋆]

[⋆]IBM Research Division
IBM Almaden Research Center
San Jose, CA 95120
{windsor,young}@almaden.ibm.com

[†]Computer Science Division
University of California
Berkeley, CA 94720
{windsorh,smith}@cs.berkeley.edu

Abstract

Recent developments in both hardware and software have made it worthwhile to consider embedding intelligence in storage to handle general purpose processing that can be offloaded from the hosts. In particular, low-cost processing power is now widely available and software can be made robust, secure and mobile. In this paper, we propose a general Smart Storage (SmartSTOR) architecture in which a processing unit that is coupled to one or more disks can be used to perform such offloaded processing. A major part of the paper is devoted to understanding the performance potential of the SmartSTOR architecture for decision support workloads since these workloads are increasingly important commercially and are known to be pushing the limits of current system designs. Our analysis suggests that there is a definite advantage in using fewer but more powerful processors, a result that bolsters the case for sharing a powerful processor among multiple disks. As for software architecture, we find that the offloading of database operations that involve only a single relation to the SmartSTORs is far less promising than the offloading of multiple-relation operations. In general, if embedding intelligence in storage is an inevitable architectural trend, we have to focus on developing parallel software systems that can effectively take advantage of the large number of processing units that will be in the system.

1 Introduction

Typical I/O devices consist of the physical device hardware (*e.g.,* disk platters, read/write heads), device specific electronics (*e.g.,* sense amplifiers) and generic electronics (a general purpose or special purpose embedded microprocessor or processors). With the rapid growth in processing power per processor (estimated at a rate of 60% per year [11]), it is reasonable to consider implementing and treating the processing power placed in a disk controller as general purpose, and not just as a dedicated microprogrammed embedded controller. For instance, a 33 MHz ARM7TDMI embedded processor has recently been used to implement all the functions of a disk controller, including the servo control [3]. If a moderately powerful general purpose microprocessor is combined with a reasonable amount of local memory, and placed either in a disk controller or a storage controller (*i.e.,* a controller which controls multiple devices), then there will exist a general purpose outboard CPU with substantial excess processing capacity.

Recent advances in software technology make using this processing capacity easier than previously. In particular, software fault isolation techniques [23] as well as robust and secure languages such as Java [9] enable applications to be effectively isolated so that they can be safely executed on a machine without causing malicious side effects. Recent emphasis on architectural neutrality and the portability of languages [9] further enhances code mobility and eases the way for code to be moved to different machines for execution. For example, in SUN's Jini framework [20], application code can be downloaded to the device as needed. The convergence of these hardware and software developments provide an opportunity for a fundamental shift in system design by allowing application code to be offloaded to the peripherals for execution.

In this paper, we propose a general architecture for *Smart Storage* in which a processing unit that is coupled to one or more disks can be used to perform general purpose processing offloaded from the host. The main motivation for SmartSTOR is that many of today's storage adapters and outboard controllers already contain several general purpose commodity processors that are needed to handle functions such as RAID [6] protection. Implementing Smart Storage would amount to enhancing these adapters and controllers to perform some general purpose processing. Besides allowing processing to be offloaded from the host processor, the Smart Storage

Most of the funding for this research has been provided by IBM Corporation; additional support has been provided by the State of California under the MICRO program, Fujitsu Microelectronics, Toshiba Corporation, Microsoft Corporation, Cirrus Corporation, Quantum Corporation and Sun Microsystems.

This report is also available as Research Report RJ 10145, IBM Almaden Research Center, San Jose, CA.

architecture also reduces data movement between the host and storage subsystem. In addition, it allows processing power to be automatically scaled with increasing storage demand. Other advantages of embedding intelligence in storage include simplifying the costly task of system management [5].

There have been some recent proposals for embedding intelligence in disks [10] and these include the Intelligent Disk (IDISK) [16] and the Active Disk [1, 18]. The processors that can be used in these disk-centric proposals are subject to the power budget and stringent cost constraints of the disk environment - generally disks are fungible and are sold almost entirely on the basis of price. The market for high cost/high performance/high functionality disks is very limited, and thus prices for disks in this market segment are higher than they would otherwise be due to the loss of efficiencies of scale. On the other hand, SmartSTOR, by operating at the level of the storage (*i.e.,* multiple device) controller, can offer processing units that are more substantial and therefore easier to effectively use. Moreover, by allowing a processing unit to be coupled to one or more disks, the SmartSTOR architecture allows for more flexible scaling of processing power to increasing storage demand. In the nearer term, the SmartSTOR architecture is likely to be easier to accomplish because increasing the processing power on an adapter or controller to handle general purpose processing is less risky than modifying the actual disk design. It also lowers the barrier of entry and opens up the architecture to the creativity of more than just the few disk companies. Finally, it separates the manufacturing of low cost disks (most of which go into PCs) from high performance controllers (which can go into servers, clusters and mainframes, and which are relatively price-insensitive).

The idea of moving processing closer to the disk was studied extensively in the form of database machines during the late 1970s and early 1980s [7, 13]. Most of those database machines relied on costly special-purpose hardware which had to be specifically programmed and which prevented the database machines from taking advantage of algorithmic advancements and improvements in commodity hardware. In addition, the reliance on highly-specialized hardware made it difficult to develop succeeding generations of the system so that it was not worthwhile to expend significant effort programming these machines. In contrast, the SmartSTOR architecture leverages commodity general purpose hardware which allows the system to track the continual improvements in both hardware and software. In particular, a SmartSTOR can be based on a standard CPU platform (*e.g.,* PowerPC, MIPS, X86, *etc.*), for which there are extensive software tools, a great deal of support, and a long projected life. In addition, the technology that is now available for developing portable and architecturally-neutral software can help reduce the need to program specifically for any particular implementation of the SmartSTOR architecture. Furthermore, shared nothing database algorithms and technology have matured to the point where we should be able to exploit some of the parallelism present in the SmartSTOR architecture.

Essentially, we envision a system in which the host supervises a number of SmartSTORs, each of which consists of a powerful processing unit, a useful amount of local memory, and a number of I/O devices, usually disks. The host processor may generate tasks specific to one SmartSTOR (*i.e.,* only needing data local to that SmartSTOR) and delegate that work to the SmartSTOR, which would then deliver the result to the host. Alternatively, the SmartSTOR can be handed more complicated tasks that require coordination with other SmartSTORs. If the generation and delegation of these tasks can be sufficiently automated and reliable, and if the load balancing is successful, then the processing power of the SmartSTOR CPUs and the host become additive, and the result is a much more powerful system.

An essential element to the success of the Smart Storage architecture lies in convincing the software developers that SmartSTOR is a viable and attractive architecture. Projecting the performance potential of the SmartSTOR architecture is an important first step in this direction. Since decision support workloads are increasingly important commercially [4], a major part of this report is devoted to understanding how these workloads will perform on the SmartSTOR architecture. In particular, we evaluate the performance of the Transaction Processing Performance Council Benchmark D (TPC-D) [21], which is the industry-standard decision support benchmark, on various SmartSTOR-based systems. Our methodology is based on projecting SmartSTOR performance from current system performance and parameters. More specifically, we use the system configurations of recent TPC-D results to determine the number of SmartSTORs that will be needed. In addition, we examine the query execution plans from two recently certified TPC-D systems to establish the fraction of work that can be offloaded to the SmartSTORs. We also use recent TPC-D results to empirically derive the system scalability relationship so that we can estimate the effectiveness of distributing a query among many SmartSTORs. There are clearly limits to this projection approach but we believe that it is the most effective and appropriate methodology at this early stage.

The rest of this report is organized as follows. In the next section, we describe the hardware and software architecture for SmartSTOR. In Section 3, we present the methodology used to project the performance of TPC-D on systems with SmartSTOR. Performance analysis results are presented in Section 4. Section 5 concludes this report. Appendix A contains a brief overview of the TPC-D benchmark whilst Appendix B contains the query execution plans of all 17 TPC-D queries taken from a recently certified TPC-D setup.

2 The SmartSTOR Architecture

The proposed Smart Storage architecture consists of a processing unit that is coupled to one or more disks. Figure 1 depicts such an architecture. We define the *cardinality* of a SmartSTOR to be the number of disks it contains. A SmartSTOR with a cardinality of one contains a *single disk* and is referred to as SD. n our performance projection, SD is conceptually equivalent to an IDISK/Active Disk. When a SmartSTOR contains *multiple disks*, we refer to it as MD.

The success of the SmartSTOR architecture hinges on the availability of software that can take advantage of its unique capabilities. Figure 2 shows a spectrum of software options, each having different performance potential and requiring different amounts of software engineering effort. At this point in time, it is not apparent which software architecture, if any, will provide enough benefits to justify its development cost but through the performance projection that we will perform later in this report, we hope to gain some understanding that will help developers reach their own conclusions.

Intuitively, data intensive operations like filtering and aggregation should be offloaded to the SmartSTOR. More generally, operations that rely solely on local data belonging to a single base relation are good candidates for offloading. We refer to this *single-relation* offloading as SR. Such operations are the basis of database queries and includes table/index scan, sort, group by and partial aggregate functions. Basically, SR includes all single-relation operations before a join or a “table queue”, which is a mechanism through which the database management system (DBMS) distributes data among its agents.

Although single-relation operations are the basis of database queries, a typical decision support query involves a lot more than

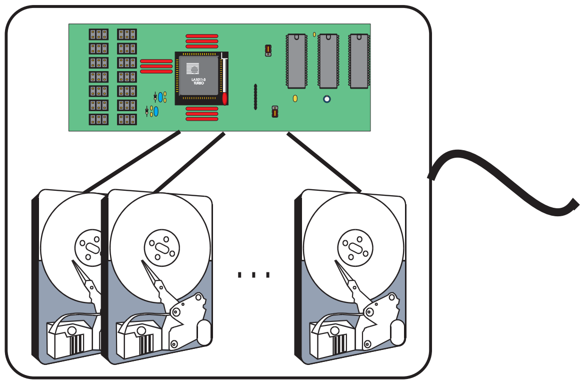


Figure 1: SmartSTOR Hardware Architecture.

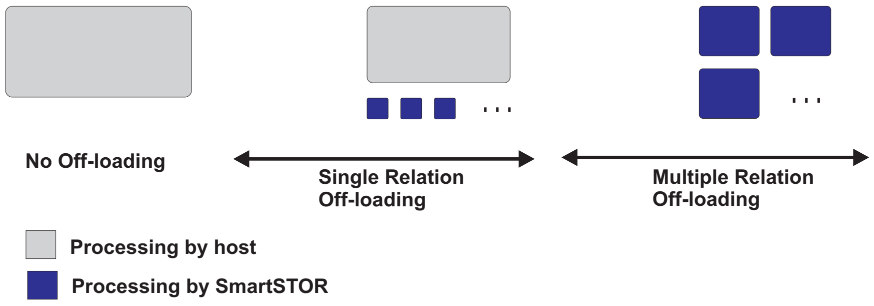


Figure 2: Possible Software Architectures.

just these basic operations. In order to distribute more processing to the SmartSTORs, we have to consider offloading multiple-relation operations such as joins that may involve data in one or more SmartSTORs. Such *multiple-relation* offloading is referred to as MR. At the extreme end, this is functionally equivalent to running a complete shared-nothing DBMS [8, 17] such as IBM's DB2/EEE [15] and an operating system on each SmartSTOR. The main shortcoming of running a shared-nothing DBMS on each SmartSTOR is the hefty resource requirement of the full-fledged DBMS. In this case, using SmartSTORs with more substantial processing unit shared among multiple disks is likely to be more effective than an IDISK/Active Disk setup. It may be possible to trim the shared-nothing DBMS to contain only the functionality profitable for offloading but coming up with this and other software architecture is an open research problem.

3 Projection Methodology

In this section, we outline the methodology that we use to assess the effectiveness of the SmartSTOR architecture and the relative merits of the various hardware and software organizations, particularly, SD (single disk), MD (multiple disk), SR (single-relation offloading) and MR (multiple-relation offloading). There have been some recent work on evaluating the performance of Active Disks [1, 2, 18, 22] but these have concentrated on image processing applications and basic database operations. Because decision support workloads represent an increasing fraction of the commercial workload [4] and are growing so rapidly as to be pushing the limits of current system designs [24, 25], we focus primarily on projecting how well they will perform on a SmartSTOR architecture. Our projection is based on the Transaction Processing Performance Council Benchmark D (TPC-D) [21], which is the industry standard benchmark for decision support. A brief description of the benchmark is provided in Appendix A. Readers who are interested in the characteristics of the benchmark are referred to [12], which contains a comprehensive analysis of the benchmark characteristics and how they compare with those of real production database workloads.

TPC-D version 2, which is substantially different from version 1, has been approved and must be used by vendors starting February 16, 1999. This report is based on version 1 since all the published results are of this version. As soon as enough TPC-D version 2 results are published, we plan to do a follow-up study to see whether the same trends are observed with the new version. Because both the hardware and software technologies are advancing rapidly, we decide to look at the more recent results, specifically those that were published between July 1998 and January 1999. We omit the very recent results because we believe that these very recent setups have been so fine-tuned for running the benchmark that attempting to lump them in with the other results would be meaningless.

For instance, Figure 3 compares the query executions times for two similar benchmark setups that were published in October 1998 and February 1999. Note that in the February 1999 run, the 17 queries have been sped up so dramatically that the two update functions (UF1 and UF2) clearly dominate the run time. We believe that such behavior results from the aggressive use of Automatic Summary Tables (ASTs) which are auxiliary tables that contain partially aggregated data. When selecting the query execution plan, the optimizer will attempt to match queries to the ASTs and perform only the necessary final aggregation. In other words, ASTs enable processing to be effectively pushed to the database load phase, which

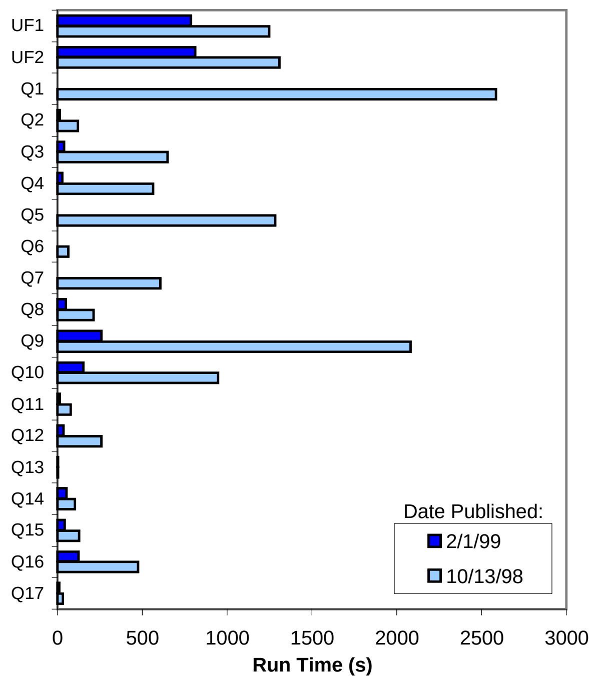


Figure 3: Profile of Query Run Times for Two Similar TPC-D Setups.

is not part of the TPC-D performance metric, so that very little processing needs to be performed when executing the queries.

3.1 I/O Bandwidth

There are two likely major advantages to the SmartSTOR architecture: (a) the amount of data that needs to be moved from the disks to the host for processing should be significantly reduced; (b) the actual processing can be offloaded from the host and done in parallel by the many processors within the whole system. Since decision support workloads are very data intensive, it is generally believed that they will benefit substantially from the potential decrease in I/O traffic. However, by considering the actual I/O bandwidth that is consumed during the execution of the TPC-D benchmark, we find that I/O bandwidth may not be that serious a bottleneck.

Based on measurements[1] performed on several certified TPC-D setups, we have been able to establish a simple rule of thumb relating the TPC-D scale factor to the amount of physical I/Os required. More specifically, we find that for a database of scale S, a total of about $3 \cdot S$ GB of data are transferred between the host and storage system during a TPC-D power test. With improvements in the memory capacity of the host system and more sophisticated database optimization, the constant 3 is expected to gradually decrease over time. Our measurements also indicate that the peak bandwidth requirement is about 3.3 times the average. Therefore, we can estimate the I/O bandwidth consumed during a TPC-D run by:

$$\begin{aligned} \textit{Average I/O bandwidth} &\approx \frac{3 \cdot S}{\textit{total run time}} \\ \textit{Peak I/O bandwidth} &\approx \frac{10 \cdot S}{\textit{total run time}} \end{aligned}$$

[1]Internal measurements taken in IBM benchmark labs.

Note that these rules of thumb are based on measurements conducted without the use of Automatic Summary Tables (ASTs).

In Table 1, we apply these rules of thumb to estimate the I/O bandwidth consumed in some recent TPC-D benchmark runs. The highest per node I/O bandwidth consumption (1251.50 MB/s peak) is observed on a 32-processor system with a 12.5 GB/s system bus and which can be configured with 32 PCI buses each having a peak bandwidth of 528 MB/s. This puts the peak bandwidth consumed at about 10% of the bandwidth available. The highest per processor I/O bandwidth consumption is about 48.43 MB/s peak and occurs on an 8-processor system with a 3.2 GB/s system bus. This system can be configured with eight 528 MB/s PCI buses. Such results suggest that decision support workloads similar to TPC-D may not impose extra I/O bandwidth burden over that required for other workloads that today's systems are designed to handle.

To further understand this rather surprising finding, let us examine the query execution plans from a recently certified TPC-D setup. These plans are presented in Appendix B. Of the 17 TPC-D queries, only Query 16 uses a table scan and it is of the SUPPLIER table which contains only about 0.1% of the total number of records in the database. All the other accesses rely on an index in one way or another. In this particular TPC-D setup, a total of twenty-six indices are defined over the eight relations. Perhaps as a reflection of the fact that the TPC-D benchmark has been well studied and understood, there are many cases of index-only-access in which all the required fields are defined in the indices. It appears that the judicious use of techniques such as indices has been extremely effective at reducing the amount of I/O bandwidth required to support a TPC-D-like decision support workload. Therefore, for the rest of this report, we will concentrate on the offloading aspect of SmartSTOR.

3.2 System Configuration

The first step in projecting the performance of TPC-D on the SmartSTOR architecture is to determine the number of SmartSTORs that will be in the system and the processing power that they will possess. As is typical of forward-looking studies, we assume that some aspect of the system, in this case the number of drives, will remain the same. Table 2 summarizes the relevant configuration information for the recent TPC-D results. For each setup, we project the number of SmartSTORs in the corresponding future system by:

$$\textit{num-SmartSTOR} = \frac{\textit{num-disk}}{\textit{cardinality}}$$

In order to describe the processing power available in the SmartSTORs without using absolute and therefore time-frame dependent numbers, we introduce the notion of *performance per disk* (perf-per-disk), which is the effective processing power per disk relative to the host processor.

$$\textit{perf-per-disk} = \frac{\textit{processing power per SmartSTOR}}{\textit{processing power of host processor} \cdot \textit{cardinality}}$$

The actual value of perf-per-disk depends on the cardinality, family and generation of processors used, the power budget, the system design, *etc.* and is open to debate. In general, we believe that if the processor is embedded in the disk as opposed to the adapter or outboard controller, it will tend to have lower performance because of the smaller power budget and the much more stringent cost constraints in the disk environment. For an intelligent adapter or controller, the embedded processor may perhaps be even as powerful as a host processor, although that would unlikely be cost effective.

	System	Average		Peak	
		MB/s per Node	MB/s per Processor	MB/s per Node	MB/s per Processor
100GB	Sun Enterprise 3500	116.22	14.53	387.40	48.43
	NEC Express 5800 HV8600	74.26	9.28	247.53	30.93
	IBM Netfinity 7000 M10	36.65	9.16	122.17	30.53
	IBM RS/6000 S70	54.97	4.58	183.23	15.27
	IBM NetFinity 7000 M10	37.10	9.27	123.67	30.90
	Compaq ProLiant 7000	41.27	10.32	137.57	34.40
	NCR 4400	24.36	6.09	81.20	20.30
	Compaq Digital Alpha 4100	20.23	5.06	67.43	16.87
300GB	IBM RS/6000 SP model 550	10.76	2.69	35.87	8.97
	Compaq Alpha Server GS140	42.75	4.27	142.50	14.23
	Sequent NUMA-Q 2000	149.78	4.68	499.27	15.60
	SGI Origin 2000	91.28	2.85	304.27	9.50
	HP 9000 V2250	70.60	4.41	235.33	14.70
	HP NetServer LXr 8000	25.42	6.36	84.73	21.20
	NCR 4400	22.84	5.71	76.13	19.03
1TB	Sun Starfire Enterprise 10000	375.45	5.87	1251.50	19.57
	IBM Netfinity 7000 M10	9.81	2.45	32.70	8.17
	Sequent NUMA-Q 2000	238.75	3.73	795.83	12.43
	Sun Starfire Enterprise 10000	281.34	4.40	937.80	14.67

Table 1: Estimated I/O Bandwidth Consumed during TPC-D.

	System	# Host Processors	# Disks	Ratio
100GB	Sun Enterprise 3500	8	138	17.25
	NEC Express 5800 HV8600	8	129	16.13
	IBM Netfinity 7000 M10	4	94	23.50
	IBM RS/6000 S70	12	215	17.92
	IBM NetFinity 7000 M10	4	84	23.50
	Compaq ProLiant 7000	4	84	21.00
	NCR 4400	4	43	10.75
	Compaq Digital Alpha 4100	4	57	14.25
300GB	IBM RS/6000 SP model 550	96	816	8.50
	Compaq Alpha Server GS140	40	512	12.80
	Sequent NUMA-Q 2000	32	263	8.22
	SGI Origin 2000	32	209	6.53
	HP 9000 V2250	16	202	12.63
	HP NetServer LXr 8000	4	89	22.25
	NCR 4400	4	63	15.75
1TB	Sun Starfire Enterprise 10000	64	1085	16.95
	IBM Netfinity 7000 M10	128	928	7.25
	Sequent NUMA-Q 2000	64	809	12.64
	Sun Starfire Enterprise 10000	64	1085	16.95
Average		31.16	363.42	14.99

Table 2: Number of Host Processors and Disks used in Recent TPC-D Setups.

Query	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
System 1	100.0	6.8	0.3	4.0	5.2	41.8	0.2	21.8	9.0	2.1	0.4	8.0	0.3	13.3	48.2	0.0	0.1
System 2	99.9	2.3	12.9	2.7	94.6	44.1	53.4	9.0	5.5	1.6	0.1	7.8	6.4	91.6	41.9	0.2	49.1

Table 3: Percent of Work that can be Offloaded by SR.

In either case, the embedded processor is likely to be used also for tasks, some of which are real-time, that are previously performed in special-purpose hardware. Since it is premature to specify precise values for perf-per-disk, we perform sensitivity analysis on the parameter in this paper.

3.3 SR Performance

Recent work has shown that single-relation operations such as SQL select and aggregation can be very effectively offloaded to an Active Disk [1]. However, a typical decision support query involves a lot more than just single-relation operations. In most cases, the results of the single-relation operations are combined through joins to create new derived relations that are further operated on. Therefore, determining the actual fraction of work that can be offloaded by SR, and thereby the potential overall speedup, is non-trivial.

Our method for determining the fraction of work that can be offloaded by SR is to analyze the query execution plans. The results presented in this paper are based on the query execution plans from two recently certified TPC-D setups. The fraction of processing that can be offloaded depends very much on the query execution plans selected by the query optimizer. In order to understand the possible range of values, we consider both a shared-everything and a shared-nothing DBMS. System 1 is a Symmetric Multiprocessor System (SMP) running IBM's DB2/UDB [14], a shared-everything DBMS, while System 2 is a cluster-based system running the shared-nothing IBM DB2/EEE [15].

Appendix B contains the plans from the first system. In our notation, any sub-tree rooted by a rectangular box is a SR sub-tree; all the operations in such a sub-tree can be offloaded by SR. Our goal is to determine the fraction of work that the SR sub-trees represent. Measuring the CPU time needed for each individual operation in a query execution plan is extremely difficult because the operations are executed simultaneously in parallel or in a pipelined fashion. Therefore, we use the CPU costs estimated by the query optimizer to determine the fraction of work that is represented by the SR sub-trees. The results for all 17 queries in the two TPC-D setups are summarized in Table 3.

From the table, Query 1 is the only query that can be offloaded by more than 50% in System 1. Observe further that only 5 out of the 17 queries can be offloaded by more than 10% in System 1. System 2 is generally more amenable towards single-relation offloading but it is still the case that less than half of the queries can be offloaded by more than 10%. According to Amdahl's Law [11], these statistics suggest that the performance potential of SR may be limited. However, the fact that there is substantial difference between the figures for the two setups suggest that there may be considerable room for improving the plans generated to better take advantage of the SmartSTOR architecture. This is an area that requires further research.

Suppose that f is the fraction of processing that can be offloaded by SR. Assuming that host and SmartSTOR processing are maximally overlapped, the speedup that can be achieved by SR is:

$$\textit{speedup} = \frac{1}{Max(1-f, \frac{f}{s})}$$

where

$$s = \frac{\textit{num-disk}}{\textit{num-host-proc}} \cdot \textit{perf-per-disk}$$

is the aggregate processing power available in the SmartSTORs relative to that in the host. If we further assume that the system will be intelligent enough to not offload operations when it does not make sense to do so, the speedup is:

$$\textit{speedup} = Max(1, \frac{1}{Max(1-f, \frac{f}{s})})$$

As we shall see, even with such optimistic assumptions, the performance potential of SR is rather limited.

Assuming that the current run time for query i is $QI(i)$, we can project the run time for the query on a SmartSTOR architecture, $QI(i)'$, by:

$$QI(i)' = \frac{QI(i)}{\textit{speedup}}$$

The TPC-D benchmark defines both a power metric and a throughput metric [21]. Since we are primarily interested in speedups, we focus on the power metric, QppD, in this paper. In determining the average performance improvement possible in a SmartSTOR architecture with SR, we use the projected query run times, $QI(i)'s$, to determine the speedup in QppD for each of the recent 19 TPC-D systems. Then we take the arithmetic mean over the 19 setups to obtain an average improvement in QppD. Note that QppD includes the execution times of two update functions, which we assume cannot be offloaded by SR. Also, as discussed in Appendix B, the definition of QppD limits the run time of any query to be at most 1000 times shorter than that of the slowest query.

3.4 MR Performance

In general, when work is distributed across multiple processing units, skew comes into play so that the performance of the system scales sublinearly with the number of processing units. For a well-understood workload such as TPC-D, we can try to distribute the tuples in the base relations evenly across the SmartSTORs so as to minimize any data skew. Therefore, for SR, the portion of work offloaded is likely to be sped up by the extra processing power available in the SmartSTORs. However, for more complicated operations that involve redistributing tuples or that involve derived relations, there is likely to be an unequal distribution of relevant tuples across the SmartSTORs.

In order to project the performance of TPC-D when multiple-relation operations are offloaded, we need to understand how effectively the work can be distributed across the SmartSTORs, *i.e.,* we need to understand the scalability of the system. Since we are not aware of any generally accepted model of scalability for TPC-D, we empirically derive a model by using the recent TPC-D results. Because these results were obtained on systems with different processors, we have to first normalize them. Let:

$$\textit{database efficiency} = \frac{\textit{QppD}}{\textit{SPECintbase95} \cdot \textit{num-host-proc}}$$

Some readers may balk at normalizing TPC-D performance by SPEC numbers [19] but we are not aware of any better alternative.

Figure 4 plots the database efficiency of the recent 300 GB TPC-D results. We choose to use the 300 GB results because the benchmark setups for this scale factor have a wide range in the number of processors used. Observe that the set of points can be roughly approximated by $\frac{C}{\sqrt[3]{\textit{num-host-proc}}}$, where C is a constant. We refer to this scalability rule as the *cube root rule* in that when the number of processors is increased by a factor of eight, the per processor efficiency is halved. We expect the scalability of the system to improve with advances in both hardware and software. Therefore, we use the *fourth root rule* to consider future TPC-D system

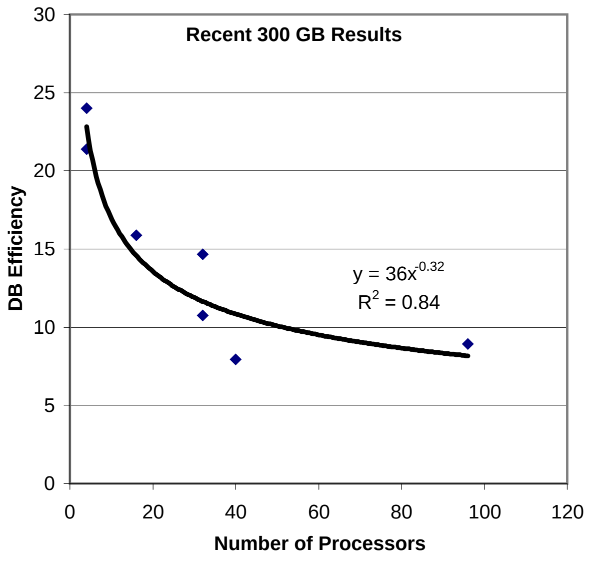


Figure 4: Scalability of TPC-D Systems.

scalability. With the fourth root rule, the per processor efficiency is halved when the number of processors is increased by a factor of 16. Note that real workloads are unlikely to be as well understood and tuned as the TPC-D benchmark and the processing will tend to be less well distributed. In other words, real workloads will probably scale more poorly with the number of processors. Therefore, we also consider the *square root rule*.

Using these scalability rules, we can establish a relationship between QppD and the number of processors and their processing power.

$$\begin{aligned} &\mathit{QppD} \\ &= \mathit{database\ efficiency} \cdot \mathit{SPECintbase95} \cdot \mathit{num\text{-}host\text{-}proc} \\ &= \frac{C}{\sqrt[n]{\mathit{num\text{-}host\text{-}proc}}} \cdot \mathit{SPECintbase95} \cdot \mathit{num\text{-}host\text{-}proc} \\ &= C \cdot \mathit{SPECintbase95} \cdot \mathit{num\text{-}host\text{-}proc}^{1-\frac{1}{n}} \end{aligned}$$

where

$$n = \begin{cases} 2 & \textit{for the square root rule,} \\ 3 & \textit{for the cube root rule,} \\ 4 & \textit{for the fourth root rule.} \end{cases}$$

In a SmartSTOR environment,

$$\begin{aligned} &\mathit{QppD} \\ &= C \cdot \mathit{SPECintbase95}_{\mathit{SmartSTOR}} \cdot \\ &\quad \mathit{num\text{-}SmartSTOR}^{1-\frac{1}{n}} \\ &= C \cdot \mathit{perf\text{-}per\text{-}disk} \cdot \mathit{cardinality} \cdot \mathit{SPECintbase95}_{\mathit{host}} \cdot \\ &\quad \mathit{num\text{-}SmartSTOR}^{1-\frac{1}{n}} \end{aligned}$$

Therefore,

$$\mathit{QppD\ improvement} = \mathit{perf\text{-}per\text{-}disk} \cdot \mathit{cardinality} \cdot \left(\frac{\mathit{num\text{-}SmartSTOR}}{\mathit{num\text{-}host\text{-}proc}}\right)^{1-\frac{1}{n}}$$

Using this result, the improvement in QppD can be projected for each of the 19 recent TPC-D systems. As in the case for SR, we take the arithmetic mean over the 19 setups to obtain the average projected improvement in QppD.

4 Analysis of Performance Results

Based on the steps outlined in the previous section, we can analytically derive the improvement in QppD for the various hardware and software alternatives. The results are summarized in Figure 5. For MR, we plot the projected range of speedup with the square, cube and fourth root scalability rules. The textured (bricked) regions in the figure are bounded by the potential speedup with the square and cube root rules. For SR, we plot the range of speedup given by the two sets of offloading fractions discussed in Section 3.2 and presented in Figure 3. Note that the figure makes no cost statement. This is deliberate since accurate cost information are generally closely guarded and in any case, are very technology and timeframe-dependent. Given a set of cost estimates, Figure 3 can be used to determine whether SmartSTOR is a cost-effective approach and if so, the configuration that should be used.

Recall from our scalability model that for MR, TPC-D performance tends to scale rather sublinearly with the number of processors used. This shows up in Figure 5 in that for the same perf-per-disk, MR4D is projected to have a performance advantage over MR2D and an even bigger advantage over MRSD. However, MD is limited by the fact that there are no arbitrarily powerful processors. A natural question to ponder at this juncture is how does IDISK/Active Disk compare with an intelligent adapter or controller? IDISK/Active Disk is conceptually identical to an intelligent adapter or controller of cardinality 1 with the exception that it is likely to have a lower perf-per-disk. As discussed earlier, the exact value of perf-per-disk is arguable but with the much more stringent power and cost constraints in the disk environment, as well as the fact that the processor may have to handle tasks previously performed in special-purpose hardware, we believe that a value of $\frac{1}{4}$ may be reasonable. For comparison, this ratio of processing power is about equivalent to that between a 200 MHz Intel Pentium MMX and a 575 MHz Compaq Alpha 21264 (based on SPECintbase95). In this case, the projected improvement in QppD ranges from 1.16 to 1.39 for SR and from 0.95 to 1.88 for MR.

As a rough guide, an interesting value of perf-per-disk for an outboard controller may be:

$$\mathit{perf\text{-}per\text{-}disk} = \frac{0.8}{\mathit{cardinality}}$$

Based on this, the projected speedup in QppD for cardinalities of 1, 2 and 4 with multiple-relation offloading ranges from 3.05 to 6.02, from 2.15 to 3.58 and from 1.52 to 2.13 respectively. For single-relation offloading, the corresponding ranges are 1.20-1.59, 1.17-1.48 and 1.15-1.35. These results suggest that the performance potential of MR is clearly superior to that of SR. An important point to note here is that among all published TPC-D results so far, the largest number of processors used is only 192 while the largest number of disks used is over 1,500. If embedding intelligence in storage is an unavoidable architectural trend, we have to focus on improving the scalability of parallel software systems to effectively take advantage of the large number of processors that will be in the system.

5 Conclusions

In this report, we have proposed a general Smart Storage (SmartSTOR) architecture in which general purpose processing can be performed by a processing unit that is shared among one or more

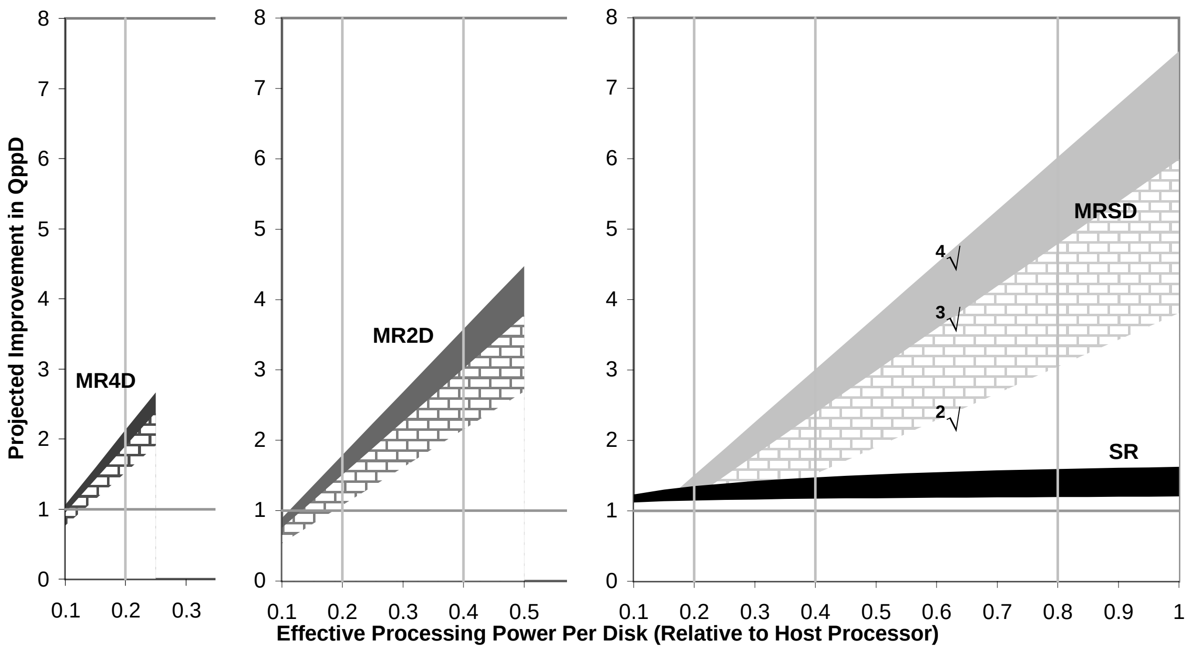


Figure 5: Projected Improvement in TPC-D Performance. To reduce clutter, we use MRSD, MR2D and MR4D to denote multiple-relation offloading on SmartSTORs of cardinality 1, 2 and 4 respectively.

disks. In order to understand the performance potential of the SmartSTOR architecture for decision support workloads, as well as the various hardware and software tradeoffs, we projected the performance of the industry-standard decision support benchmark, TPC-D, on various SmartSTOR-based systems. In particular, we performed measurements on several recently certified TPC-D systems to estimate the I/O bandwidth required for supporting such workloads. We also examined the query execution plans from two recent TPC-D systems to determine the amount of processing that can potentially be offloaded to the SmartSTORs. In addition, we analyzed recent TPC-D performance figures to empirically establish a scalability rule that can be used to project the effectiveness of distributing query execution among a large number of SmartSTORs.

The SmartSTOR architecture provides two key performance advantages, namely a reduction in I/O movement between the host and I/O subsystem, and the ability to offload some of the work from the host processor to the processing units in the SmartSTORs. The analysis performed in this paper suggests that I/O bandwidth may not be that serious a bottleneck for TPC-D. Therefore the main advantage of using SmartSTORs for workloads similar to TPC-D appears to be the ability to offload some of the processing from the host. By analyzing recent TPC-D results, we find that the performance of decision support systems scales rather sublinearly with the number of processors used. Therefore, our results indicate that there is a definite advantage in using fewer but more powerful processors. In view of this and the arguments presented in the paper, we believe that intelligent adapters or controllers that share a substantial processing unit among multiple disks may be an interesting architecture. As for software architecture, our evaluation shows that the offloading of database operations that involve multiple relations is far more promising that the offloading of operations that involve only a single relation. In either case, if embedding intelligence in storage is an inevitable architectural trend, we have to develop parallel software systems that are more scalable so as to effectively take advantage of the large number of processing units that will be in the system.

Acknowledgments

We would like to thank Guy Lohman for providing the query execution plans used in this study. We would also like to thank Joseph Hellerstein, Kimberly Keeton, Bruce Lindsay and Jai Menon for reviewing early versions of this paper.

References

[1] A. Acharya, M. Uysal, and J. Saltz, “Active disks: Programming model, algorithms and evaluation,” in *Proc. Eighth Intl Conf. on Architectural Support for Programming Languages and Operating Systems*.

[2] A. Acharya, M. Uysal, and J. Saltz, “Structure and performance of decision support algorithms on active disks,” Technical Report TRCS98-28, Computer Science Department, University of California, Santa Barbara., Nov. 2, 1998.

[3] L. Adams and M. Ou, “Processor integration in a disk controller: Embedding a RISC processor in a complex ASIC to reduce cost and improve performance,” *IEEE Micro*, vol. 17, no. 4, pp. 44–48, July/Aug. 1997.

[4] P. Bernstein, “Database Technology: What’s Coming Next?” Keynote speech at *Fourth Symposium on High Performance Computer Architecture*, Feb. 1998.

[5] A. Brown, D. Oppenheimer, K. Keeton, R. Thomas, J. Kubiatowicz, and D. A. Patterson, “ISTORE: Introspective storage for data-intensive network services,” Technical

Report CSD-98-1030, Computer Science Division, University of California, Berkeley, Dec. 23, 1998.

[6] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: high-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145–185, June 1994.

[7] D. J. DeWitt and P. B. Hawthorn, "A performance evaluation of database machine architectures," in *Proc. 7th Intl Conf. Very Large Data Bases*, Sept. 1981.

[8] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H. i Hsiao, and R. Rasmusen, "The gamma database machine project," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 1, pp. 44–62, Mar. 1990.

[9] J. Gosling and H. McGilton, "The Java^{TM} language environment a white paper," May 1996. http://java.sun.com/docs/white/langenv.

[10] J. Gray, "Put EVERYTHING in the storage device." Talk at NASD Workshop on Storage Embedded Computing, June 1998. http://www.nsic.org/nasd/1998-jun/gray.pdf.

[11] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach*. Morgan Kaufmann Publishers, Inc. San Francisco, CA, second ed., 1996.

[12] W. W. Hsu, A. J. Smith, and H. C. Young, "I/O characteristics of production database workloads and the TPC benchmarks - a comparative analysis at the logical level," 1999. In preparation.

[13] A. R. Hurson, L. L. Miller, and S. H. Pakzad, *Parallel Architectures for Database Systems*. CS Press Tutorial, 1989.

[14] IBM Corporation, *DB2 UDB V5 Administration Guide*. 1997.

[15] IBM Corporation, *IBM DB2 Universal Database Extended Enterprse Edition for UNIX Quick Beginnings Version 5*. 1998.

[16] K. Keeton, D. Patterson, and J. Hellerstein, "A case for intelligent disks (IDISKs)," *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 27, no. 3, pp. 42–52, 1998.

[17] R. A. Lorie, J.-J. Daudenarde, J. W. Stamos, and H. C. Young, "Exploiting database parallelism in a message-passing multiprocessor," *IBM Journal of Research and Development*, vol. 35, no. 5/6, pp. 681–695, Sept./Nov. 1991.

[18] E. Riedel, G. A. Gibson, and C. Faloutsos, "Active storage for large-scale data mining and multimedia," in *Proc. 24th Intl Conf. Very Large Data Bases*, pp. 62–73, Aug. 1998.

[19] Standard Performance Evaluation Corporation, "SPEC CPU95 benchmarks," Aug. 1995. http://www.spec.org/osg/cpu95.

[20] Sun Microsystems, "Jini^{TM} technology architectural overview," Jan. 1999. http://www.sun.com/jini/whitepapers/architecture.html.

[21] Transaction Processing Performance Council, *TPC Benchmark^{TM} D Standard Specification Revision 1.3.1*. Dec. 1997.

[22] M. Uysal, A. Acharya, and J. Saltz, "An evaluation of architectural alternatives for rapidly growing datasets: Active disks, clusters, SMPs," Technical Report TRCS98-27, Computer Science Department, University of California, Santa Barbara., Nov. 2, 1998.

[23] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham, "Efficient software-based fault isolation," in *Proc. 14th ACM Symposium on Operating Systems Principles*, pp. 203–216, 1993.

[24] R. Winter and K. Auerbach, "Giants walk the earth: the 1997 VLDB survey," *Database Programming and Design*, vol. 10, no. 9, Sept. 1997.

[25] R. Winter and K. Auerbach, "The big time: the 1998 VLDB survey," *Database Programming and Design*, vol. 11, no. 8, Aug. 1998.

Appendix A

The Transaction Processing Performance Council Benchmark D (TPC-D) [21] is a decision support benchmark that models the analysis end of the business environment where trends are analyzed and refined to support sound business decisions. It consists of 8 relations, 17 read-only queries and 2 update functions. The 17 read-only queries have different complexities, varying from single table aggregation (*e.g.,* Query 1) to 8-way join (*e.g.,* Query 2).

Eight *scale factors* (SF) are defined – 1, 10, 30, 100, 300, 1,000, 3,000, and 10,000. The scale factor is approximately the logical database size measured in GBs. Each benchmark configuration may define different indices. With index and database storage overhead (*e.g.,* free space), the actual database size may be much bigger than the raw database size defined by the benchmark. Only results measured against the same database size are comparable.

TPC-D introduces two performance metrics and a single price-performance metric. They are the TPC-D *power metric* (QppD@Size), TPC-D *throughput metric* (QthD@Size) and TPC-D *price/performance metric* (Price-per-QphD@Size). The power metric is defined as follows:

$$QppD@Size = \frac{3600}{\sqrt[19]{(RI(1) * RI(2) * ... * RI(17) * UI(1) * UI(2))}} * SF$$

where

- RI(i) = MAX(QI(i), $\frac{1}{1000}$ MAXQI).
- QI(i) is the run time, in seconds, of query Q_i during the power test.
- MAXQI = MAX(QI(1), QI(2), ..., QI(17)).
- UI(j) is the run time, in seconds, of update function UF_j during the power test.
- Size is the database size chosen for the measurement and SF, the corresponding scale factor.

The power test runs one query at a time in the order defined by the benchmark. The 3600 translates QppD to a query per hour measurement. Since QppD is a geometric mean of query rates, each query or update function has an equal weight. If the performance of any query or update function is improved by a factor of 2, the QppD@Size measurement will be increased by about 3.7%.

If a system's execution time scales linearly with SF, QppD at any database size will be the same.

In the throughput test, one or more query streams are run concurrently on the system. The throughput metric is defined as follows:

$$\textit{QthD@Size} = \frac{S * 17 * 3600}{T_s} * SF$$

where

- S is the number of query streams used in the throughput test.
- T_s is the interval, in seconds, between when the query streams are started and when the last query stream completes.
- Size is the same as in the definition of QppD.

Notice that QthD@Size is based on the arithmetic mean of the query execution times. Thus queries with longer execution times have more weight in the metric.

The TPC-D power metric and the TPC-D throughput metric are combined to form a composite query-per-hour rating, QphD@Size, which is the geometric mean of QppD@Size and QthD@Size. Finally, the price/performance metric is defined as:

$$\textit{Price-per-QphD@Size} = \frac{\$}{\textit{QphD@Size}}$$

Appendix B

In this appendix, we show the query execution plans of all 17 TPC-D read-only queries from a recently certified TPC-D result on an SMP system. In each execution plan, sub-trees rooted by rectangular boxes are SR sub-trees; all operations in a SR sub-tree are single-base-relation operations that can be offloaded by SR. Each query execution plan tree is rooted by a *return* operation, which returns the qualified tuples to the application. We use a double-circle for tables and indices. These include base tables, base indices, and optimizer generated subqueries and table functions. The legend is as follows:

- AST: automatic summary table
  ASTs are auxiliary tables that contain partially aggregated data.
- FETCH: table scan through index
- FILTER: predicate evaluation
  This is for the predicates that are not pushed down to the scans.
- [HS|MS|NL]JOIN: [hash|merge-scan|nested-loop] join
- [IX|TB]SCAN: [index|table] scan
  IXSCAN is different from FETCH in that the former is an index-only scan, *i.e.*, the corresponding table does not have to be read to get the needed fields.
- L[M]TQ: local [merge] table queue
  Table queue is a mechanism to exchange data among operations. The plans are from an SMP system, thus, all table queues are local table queues. Regular LTQ collects data in any order while LMTQ collects data in a specific order.
- GRPBY: group by
- RETURN: return to host
- SORT: sort

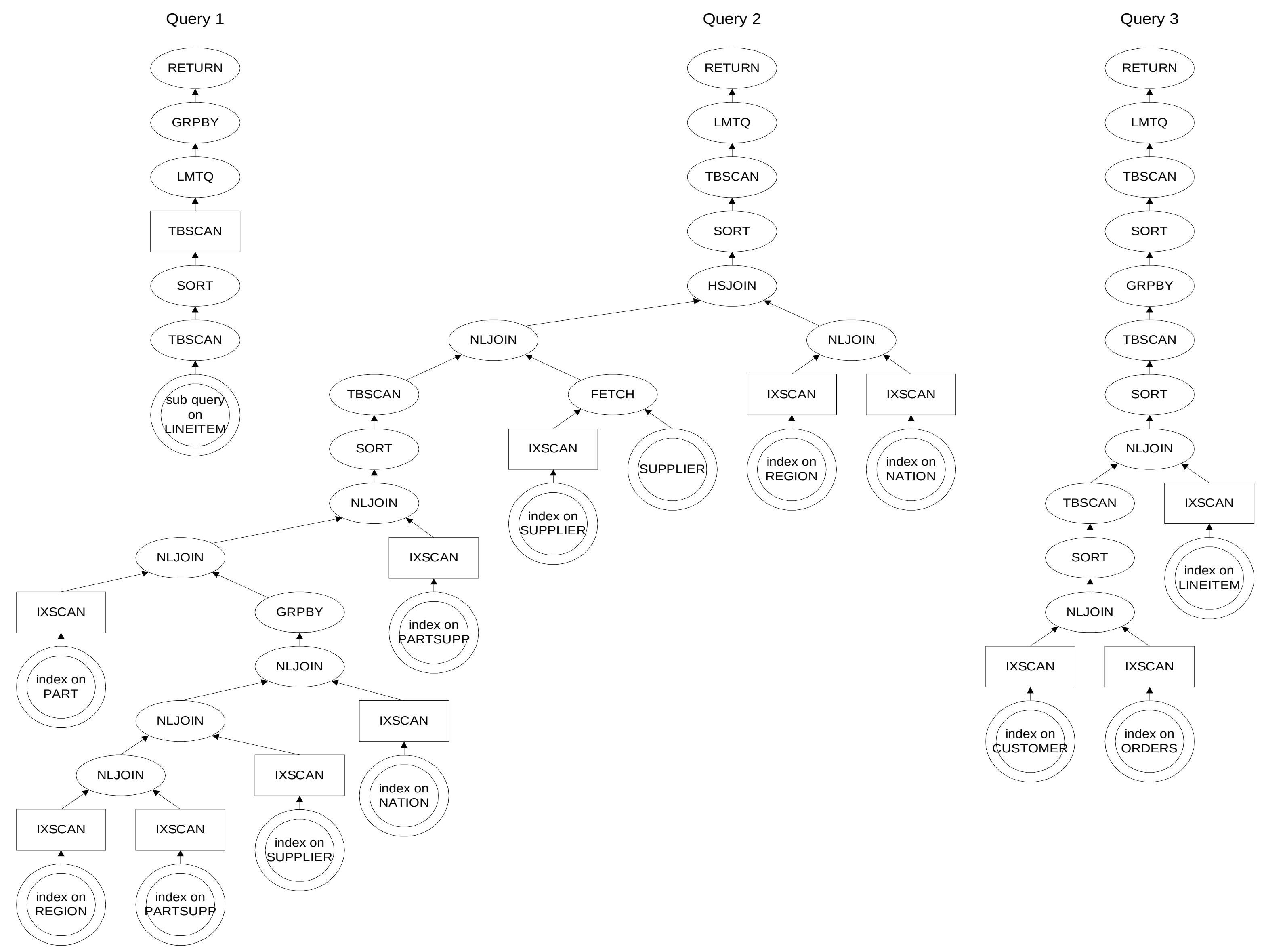


Figure B-1: Execution Plans for Queries 1, 2 and 3.

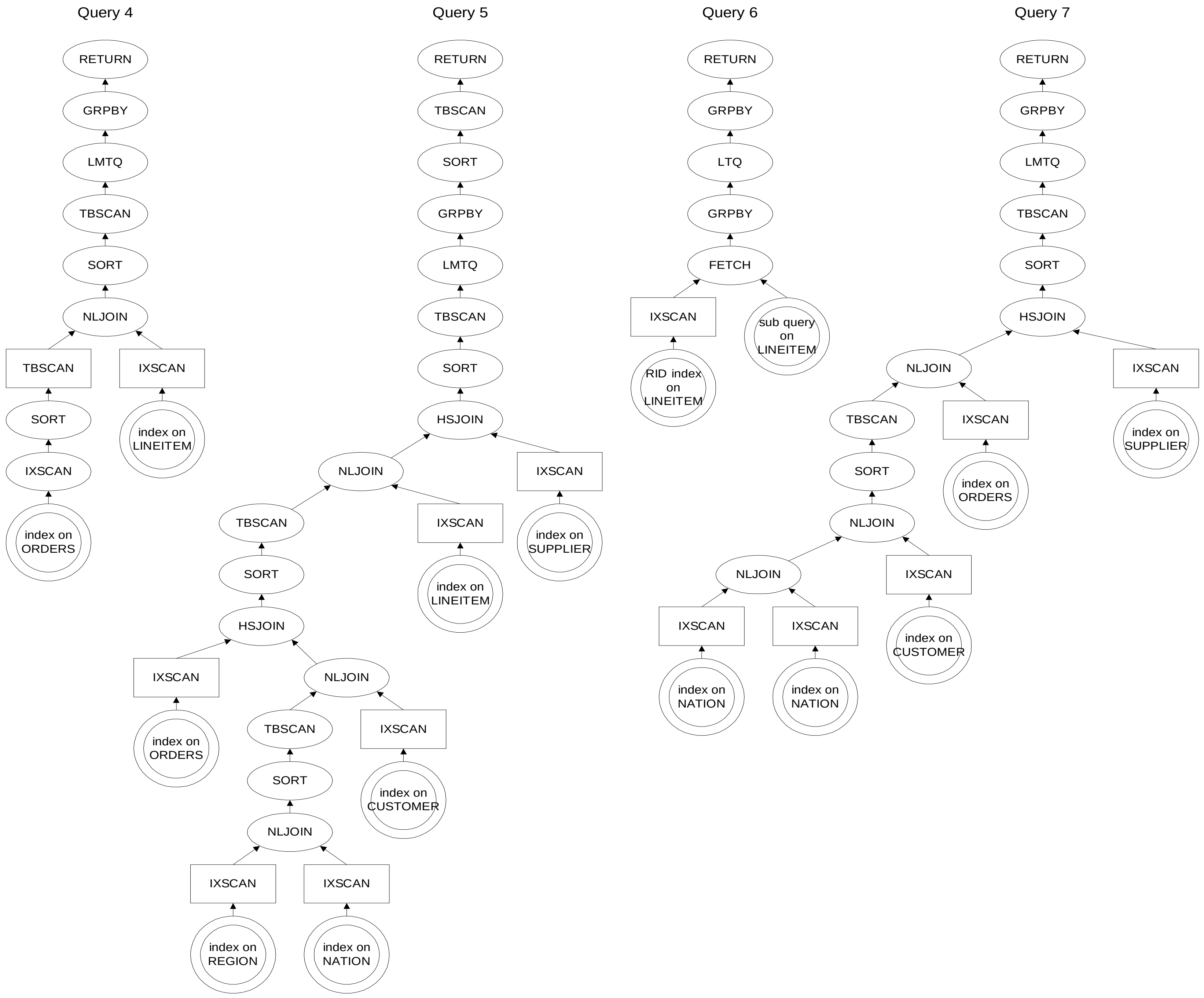


Figure B-2: Execution Plans for Queries 4, 5, 6 and 7.

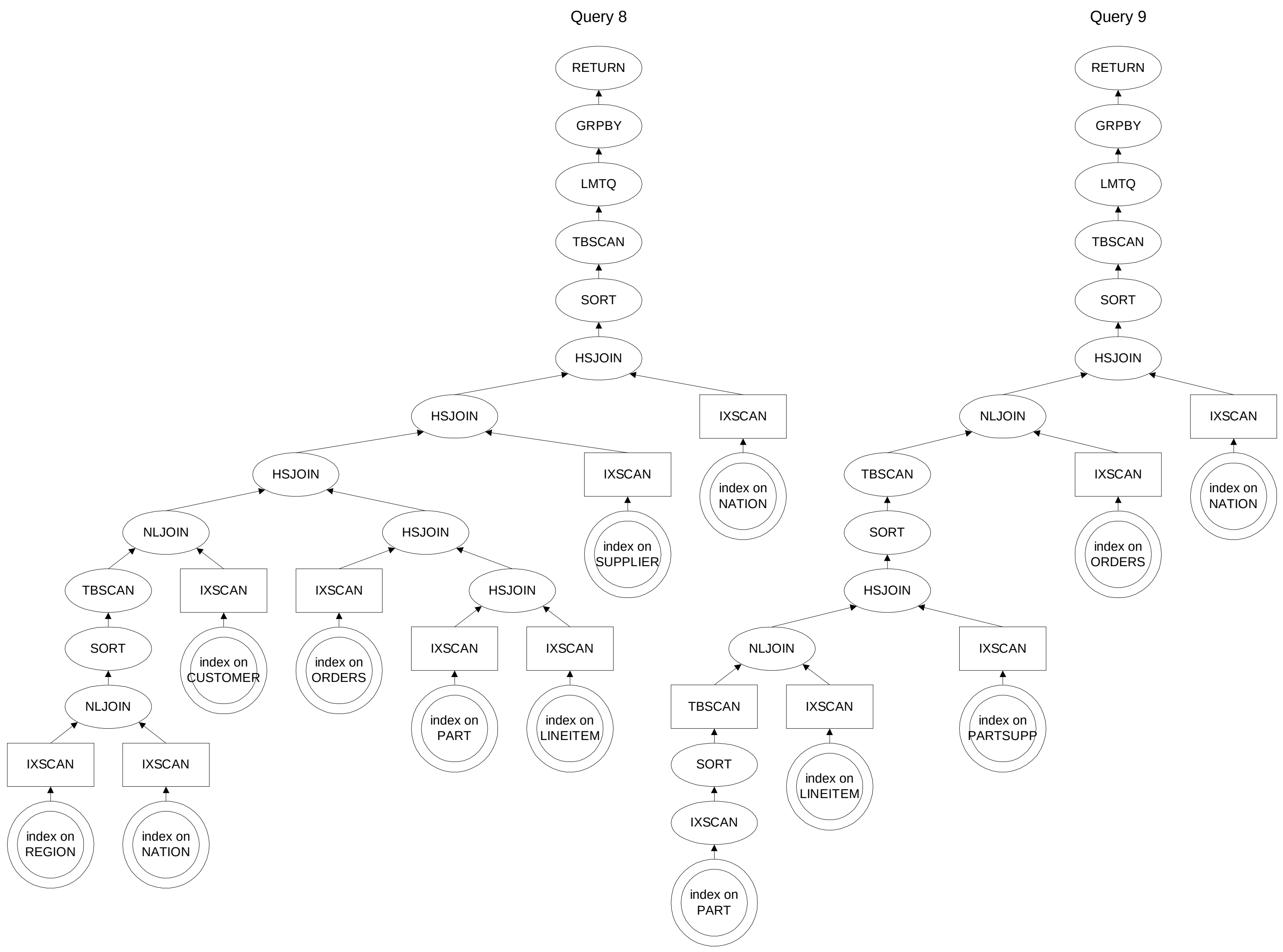


Figure B-3: Execution Plans for Queries 8 and 9.

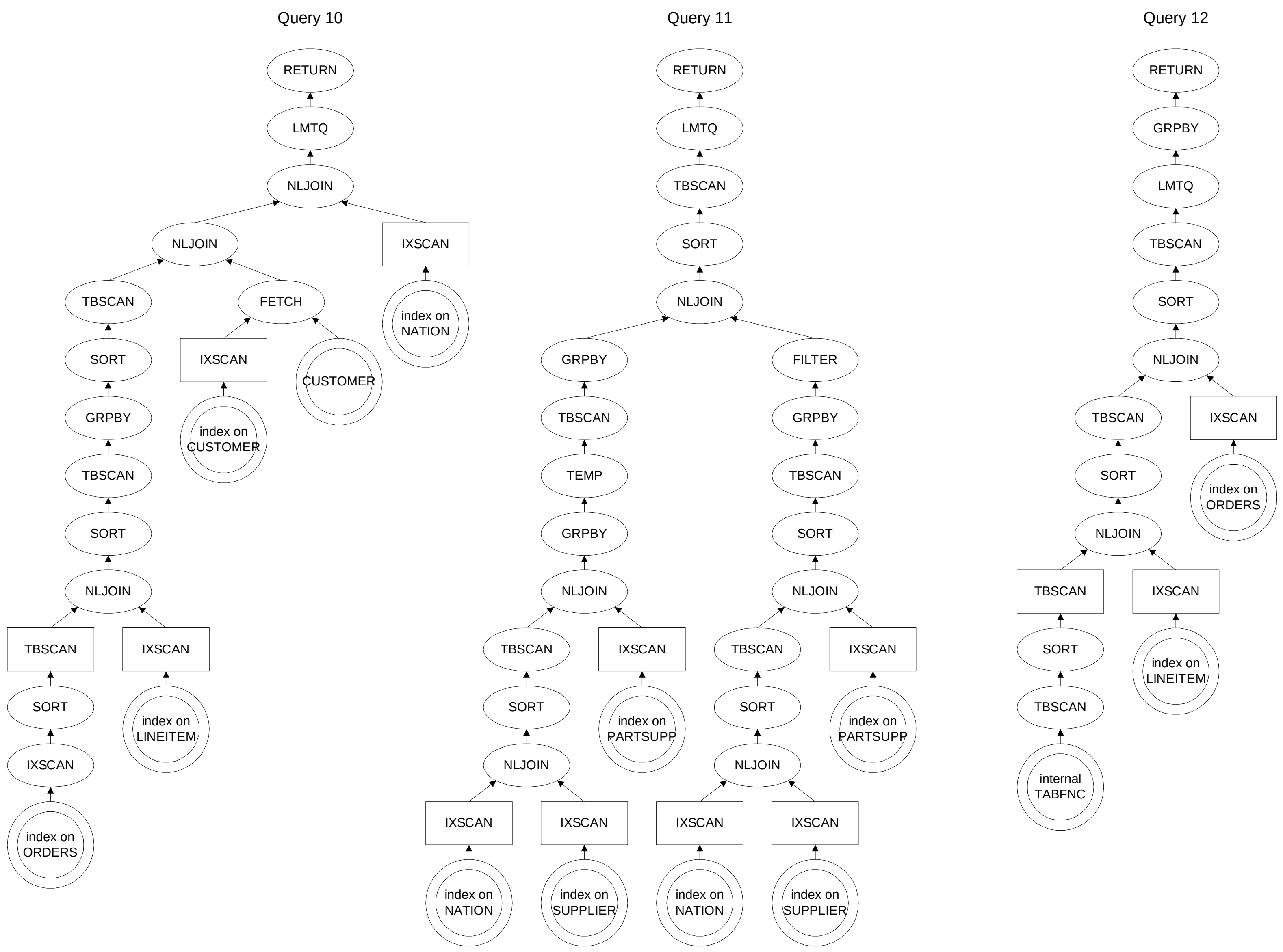


Figure B-4: Execution Plan for Queries 10, 11 and 12.

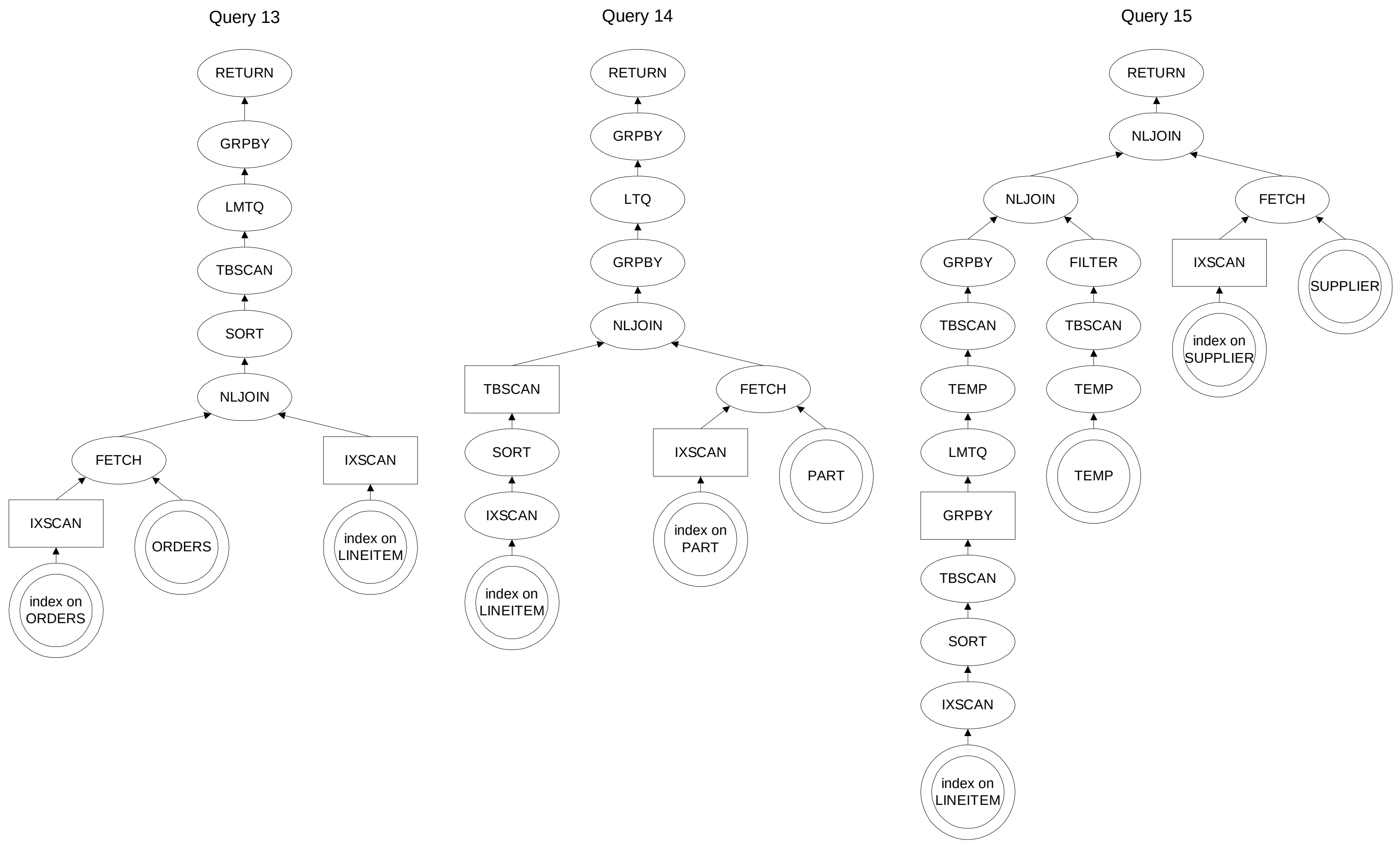


Figure B-5: Execution Plans for Queries 13, 14 and 15.

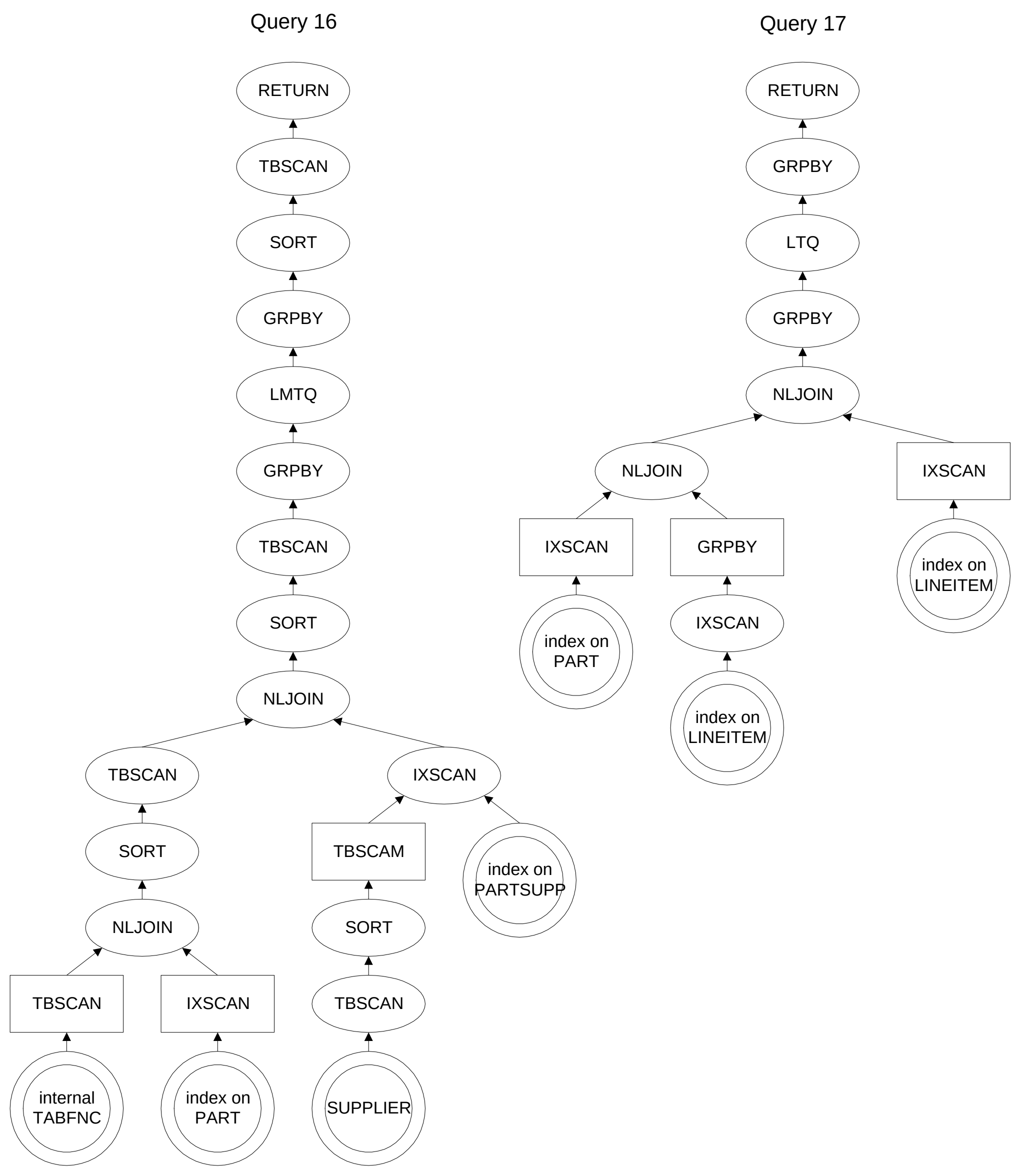


Figure B-6: Execution Plans for Queries 16 and 17.