Performance Analysis of Decision Support Workloads for the Desktop Environment

Swaroop Kavalanekar, Sohum Sohoni, Yiming Hu
Technical Report
Dept. of ECECS
University of Cincinnati
Cincinnati, OH 45221-0030
e-mail: {skavalan, ssohoni, yhu}@ececs.uc.edu

Abstract

Database applications are one of the fastest-growing classes of applications. Their execution characteristics differ from those of scientific or general-purpose applications. Due to their proprietary nature, software licenses typically prevent the disclosure of performance information. This has resulted in limited research being carried out in analyzing their performance.

Improvements in technology have resulted in increases in processor speed and decreases in cost of main memory and I/O latency. This has resulted in more powerful desktops systems being available at lower prices. In the near future, it is likely that desktop environments may be the choice for small to medium sized database applications.

We analyze the performance of Decision Support Systems for the desktop environment. We implement the TPC-H DSS benchmark using the SHORE storage manager on the Windows 2000 platform for this purpose. We find that there is a high amount of sequentiality in the accesses to the database. The file cache performance is very good with copy read hit rates of more than 95 percent. We also find that the performance and scalability of the database depends on the nature of the queries as well as the size and content of the database. Based on our observations, we believe that the desktop environment can support small to medium scale database applications.

1 Introduction

The phenomenal growth of transistor densities has resulted in the development of extremely fast processors and low-cost high-capacity DRAM modules. These developments have led to powerful desktop systems becoming available at lower costs. This leads us to believe that in the near future powerful desktop systems might be used for applications such as small to medium database systems. However, such desktop systems are not specifically optimized for database applications. They use off-the-shelf hardware and software components and operating systems that are tuned for general-purpose applications.

Database systems typically employ complex locking mechanisms and access large amounts of data. Thus they have different characteristics compared to general purpose and scientific applications. Due to the proprietary nature of most database systems, software licenses typically prevent the disclosure of performance information. Hence, not much research has been done on analyzing the characteristics of database systems. Most of this previous research has focused on analyzing the performance of large-scale database systems for high-end servers and for various models of distributed systems. To the best of our knowledge there has been no published work on the analysis of database workloads on desktop systems.

This work involves the performance analysis of database applications, specifically decision support workloads, for the desktop environment. Decision Support Systems (DSS) form an important class of database applications that are used to support the making of business decisions. They typically compute business trends through complex queries that access large amounts of data. We used the TPC-H benchmark [8], an industry standard Decision Support Benchmark, in our work.

The goal of our performance analysis was to observe the performance of general-purpose desktop systems for DSS workloads and identify bottlenecks, if any. Since neither the hardware nor the operating system was optimized for database workloads we studied the entire system-wide characteristics. We analyzed the performance of the processor, the memory hierarchy and the operating system.

The remainder of this paper is organized as follows. This section gives the motivation for our work. Section 2 describes the related work. Section 3 describes the TPC-H Decision Support Benchmark. Section 4 describes the SHORE storage manager. We describe our methodology in Section 5 and the results in Section 6. Section 7 gives a summary of our analysis.

2 Related Work

In this section we look at related research carried out in the area of analyzing database applications and improving their performance.

Trancoso et al. [17] analyze the memory performance of the TPC-D DSS workload for shared-memory multiprocessors. They simulate a 20 MB database using the Postgres95 database system interfaced with Mint, an execution-driven simulator. They observe that queries using indices suffer most of their misses on lock-related data and metadata, whereas queries that sequentially scan the database tables suffer most of their misses while accessing the database records. They note that these queries exhibit spatial locality, which can be exploited by using longer cache lines and through data prefetching.

Zhang et al. [12] study the interaction between the database and the operating system. They use the TPC-H benchmark as the database workload on a Linux cluster. Their analysis shows the dominance of the I/O subsystem for query execution. They show that the overhead of I/O is not only because of disk latencies but also due to the implementation of the *pread* system call for the Linux operating system. They also note that network communication is not a dominant issue for small clusters but it may be more important for larger clusters.

Barroso et al. [4] study the memory subsystem, particularly the processor cache, for the TPC-B and TPC-D database workloads and World Wide Web index search workloads. They suggest that the biggest improvements in the memory hierarchy are likely to come from larger first level caches. They perform simulation studies using the SimOS system simulator to analyze the effects of changing the processor cache parameters. They show that while the Online Transaction Processing (OLTP) workload has significant instruction and data locality that can be effectively captured by large off-chip caches, the DSS and the Web workloads are primarily sensitive to the size and latency of on-chip caches.

Karlsson et al. [14] present an analytical model to study how working sets scale with database size and other application parameters for Decision Support Systems. They show that the performance-critical working set is small and comprises of instructions and private data that are required to access a single tuple. They also show that the temporal locality exhibited by database data depends on the structure of the query and the size and contents of the database.

Ailamaki et al. [2] study the query execution time framework for four commercial database systems. To identify common trends across the database systems, they select a workload consisting of range selection and joins running on a memory resident database. They conclude that most of the query execution time is spent on stalls. They emphasize the need to focus on the data layout of the second level cache.They show that first level instruction cache misses dominate memory stalls and suggest optimizing the critical path for the instruction cache.

The above work differs from our work in the following aspects. We evaluate the performance of database applications for the desktop environment. We carry out a system-wide analysis of the TPC-H Decision Support benchmark. We believe that a small database size does not present a true picture of system utilization. Hence we use a realistic database size of 1 GB for our analysis. We try to get a higher-level view of the interaction of database workloads with the various system components such as the processor, memory and disk in a desktop environment. Based on our analysis we make recommendations for their performance improvement.

3 The TPC-H Benchmark

The TPC-H benchmark is an industry-standard decision support benchmark provided by the Transaction Processing Performance Council (TPC). TPC-H represents a typical decision support environment. It uses a generic model of any

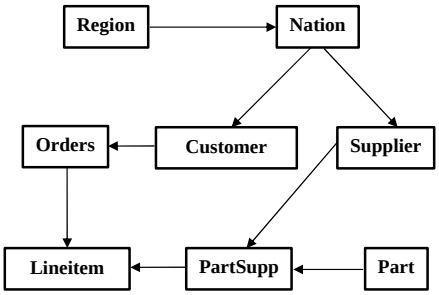

Figure 1: The TPC-H Schema [8]

industry that needs to manage, sell and distribute products world-wide. It consists of 22 read-only queries and 2 refresh functions.

We used dbgen, a database population generator, provided with TPC-H, to create the database. Figure 1, reproduced from the TPC-H standard specification revision 1.3.0 [8], illustrates the entity relationship (ER) diagram of this schema. The TPC-H Schema consists of eight base tables in 3rd normal form that contain the data for the simulated company. The tables *Nation, Region, Part, Supplier, Customer, and Orders* contain the relevant data as indicated by their names. The table *PartSupp* contains information about parts and suppliers such as the availabe quantity of a given part with a given supplier. Details of the ordered parts are stored in the table *Lineitem*. The tables *Lineitem* and *Orders* contain more than 80 percent of the total data.

```
select
        nation,
        oyear,
        sum (amount) as sum_profit
from (
        select
                n_name as nation,
                extract (year from o_orderdate) as o_year,
                l_extendedprice * (1 – l_discount) – ps_supplycost * l_quantity as amount
        from
                part,
                supplier,
                lineitem,
                partsupp,
                orders,
                nation
        where
                s_suppkey = l_suppkey
                and ps_suppkey = l_suppkey
                and ps_partkey = l_partkey
                and p_partkey = l_partkey
                and o_orderkey = l_orderkey
                and s_nationkey = n_nationkey
                and p_name like '%[COLOR]%'
        ) as profit
group by
        nation
        o_year
order by
        nation
        o_year desc;
```

Figure 2: Query 9: Functional Query Definition

We use Query 9, *Product Type Profit Measure Query*, as a running example to illustrate our implementation and results in the further sections. Query 9 determines how much profit is made on a given line of parts, broken down by supplier, nation and year. Figure 2 shows the functional query definition of Query 9 that we reproduce here from the TPC-H standard specification [8] for convenience. Figure 3 shows the query execution plan that we created for Query 9. When generating the query execution plan we implemented the operations with the most restrictive condition first. This ensures that we do not make redundant accesses to the relations in the database. It also reduces the size of the temporary

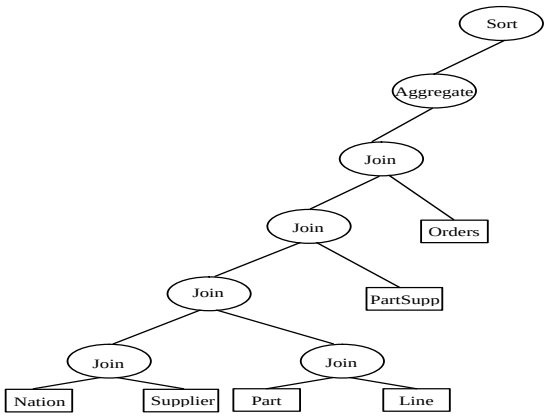

Figure 3: Query 9: Query Execution Plan

relations generated during query execution.

	System A	**System B**	**SystemC**	**System D**
Processor Speed	Pentium 4 1800 MHz	Pentium 4 1800 MHz	Pentium 3 800 MHz	Pentium 4 1500 MHz
L1 Cache	8KB 4-way associative Data Cache with 64-byte line size and 12 KB Instruction Trace Cache	8KB 4-way associative Data Cache with 64-byte line size and 12 KB Instruction Trace Cache	16 KB Data Cache and 16 KB Instruction Cache	8 KB 4-way associative Data Cache with 64-byte line size and 12 KB Instruction Trace Cache
L2 Cache	512 KB 8-way associative Unified Cache with 128-byte line size	512 KB 8-way associative Unified Cache with 128-byte line size	256 KB 8-way associative Unified Cache with 64-byte line size	512 KB 8-way associative Unified Cache with 128-byte line size
Front Side Bus	100 MHz Quad-Pumped	100 MHz Quad-Pumped	133 Mhz	400 MHz
Main Memory	256 MB RDRAM	512 MB RDRAM	512 MB SDRAM	512 MB SDRAM
Disk	60 GB IDE IBM	60 GB IDE IBM	80 GB IDE IBM	80 GB IDE IBM
Operating System	Windows 2000 SP2	Windows 2000 SP2	Windows 2000 SP2	Windows 2000 SP2

Table 1: System Configurations

4 SHORE

SHORE (Scalable Heterogeneous Object Repository) is a persistent object system developed at the University of Wisconsin [5]. It represents a merger of file system technologies and object oriented databases. It provides an object namespace similar to that of UNIX. It provides concurrency control via locking at various granularity levels. It also supports crash recovery through logging and check-pointing. SHORE has been used extensively in the database research community as well as in some commercial database systems [1, 3, 10, 11].

The SHORE server consists of two components: the SHORE storage manager (SSM) and the Value Added Server (VAS). The SSM is primarily responsible for managing the persistent object store, and for providing core services such as disk and buffer management, transactions, concurrency control and recovery. The VAS is used to extend the services

provided by the SSM. Our Value Added Server performs two primary functions. First, it provides an interface to the SSM for accessing the data stored in it. Second, it is used to implement the queries of the TPC-H benchmark.

5 METHODOLOGY

In this section we describe Intel's VTune Performance Analyzer [6], the tool we use for our analysis. This is followed by a brief description of our experimental setup and our query classification.

5.1 VTUNE Performance Analyzer

We used Intel's VTune Performance Analyzer for a comprehensive performance analysis of the TPC-H workload. VTune helps determine how system resources such as the processor, caches, memory and disks are utilized. It uses hardware counters provided by the Pentium processor and software counters provided by the Windows NT/2000 operating systems. We used the Microsoft TechNet Guide [7] to gain detailed knowledge of the software counters and what their values indicate. The counter details are provided along with the results in Section 6.

5.2 Experimental Setup

To study the effects of varying parameters such as processor speed and memory size on the overall system performance, we analyzed four systems. Table 1 shows the configurations for the four systems.

We see that System A and System B have similar configurations, with the only difference being the memory size. A comparison of System A and System B shows the effects of changing the memory size. System D has similar system parameters as System C with the only change being the processor. A comparison of System C and System D shows the effects of changing the processor speed. Since the processor architecture is different for the processors of System C and System D, (System C has a Pentium 3 processor whereas System D has a Pentium 4 processor) a direct comparison cannot be made between the two processors based on the processor speeds alone. Hence, we run CPUmark [13], a CPU intensive benchmark, on the two systems. CPUmark measures the performance of the processor subsystem by running a test workload that has been developed from profiling data obtained from major processor vendors such as Intel and AMD. We find that System D runs 23.2 percent faster than System C.

5.3 Query Classification

Figures 4 and 5 show the query execution times for various database sizes on a test system similar to System B, with 640 MB RAM. Note that in Figure 5 we eliminate the top portion of the bars for Query 5 in order to better show the differences between the other queries. Query 5 takes substantially more time as compared to the other queries in this category. We have grouped the queries into two categories, linear and exponential. As we see from Figure 4 the execution time for the queries in the linear category increases relatively linearly with an increase in the database size. On the other hand, we see from Figure 5 that the execution time for some queries increases exponentially.

This difference in behavior can be attributed to the various operations involved and the size of the input relations for those operations. Thus the nature of the queries and the database determines the size of the input and intermediate relations. The categorization is also dependent on the system parameters. Changes in the system parameters such as changing the memory size so that the working set no longer fits in the memory can cause a query to move from one category to another. Our goal in making this classification is to enable us to select representative queries for analysis.

We chose three queries for our analysis: Query 6, Query 8 and Query 9. Query 6 is a simple query. The most significant operation it performs is a sequential scan of the line relation. It does not have any join or grouping operations. Since scanning the input relations is an essential component of all other operations, we chose this query. We chose one more query from each category: Query 8 from the group of linear queries, and Query 9 from the group of exponential queries.

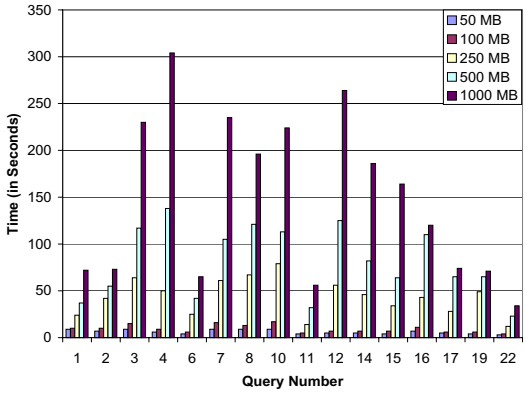

Figure 4: Linear Queries

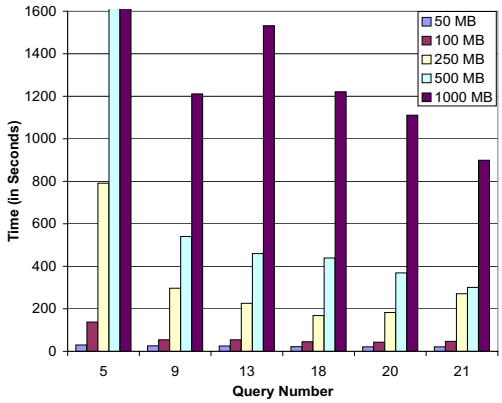

Figure 5: Exponential Queries

6 Results

In this section we present the results of our experiments. We start with a comparison of the execution times and processor utilization for the three representative queries on all four systems followed by the analysis of the memory hierarchy.

6.1 Comparison of Query Execution Time

Figure 6 shows the speedup of System B over System A in terms of the execution time. Speedup is calculated as follows.

$$Speedup(n) = \frac{Execution\,Time\,on\,A}{Execution\,Time\,on\,B}$$

We find that the execution speed of the queries improves across all queries when we compare System A with System B. The improvement is largest for Query 9 and smallest for Query 6. We shall see later that 256 MB of memory in System A is insufficient for its execution. We also see that Query 6 shows marginal improvement with an increase in memory, since 256 MB is sufficient for this query on these systems.

Figure 7 shows the average processor utilization during the execution of the three queries on System A and System B. We see that for Query 8 and Query 9, System B has higher CPU utilization than System A. Since System B has

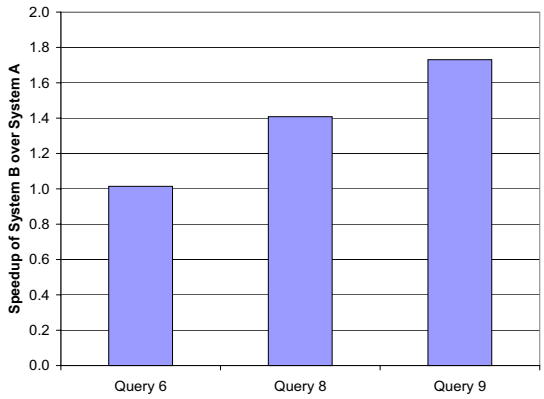

Figure 6: Comparison of Query Execution Time on System A and System B

more memory than System A, there is more data available for processing on System B. Hence, its processor is better utilized. For Query 6 we see that the CPU utilization is nearly the same for the two systems as increasing memory does not significantly affect the performance of Query 6. This leads us to believe that increasing memory alone may not lead to improved performance. We suggest increasing data prefetching to further improve performance for such queries.

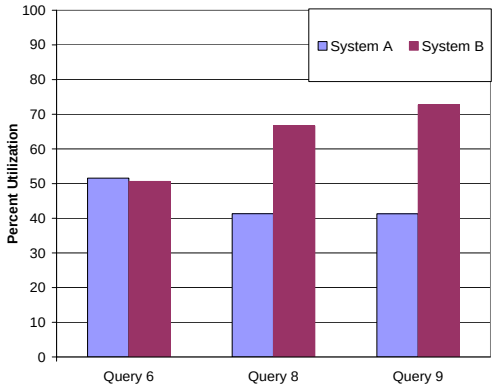

Figure 7: Processor Utilization: System A and System B

Figure 8 shows the speedup of System D over System C for the three queries. Speedup is calculated as follows.

$$Speedup(n) = \frac{Execution\,Time\,on\,C}{Execution\,Time\,on\,D}$$

As we have seen in Section 5.2, the processor subsystem of System D is approximately 23 percent faster than that of System C. Since the size of the memory is the same for both the systems and the working set of the queries fit in the memory, the queries execute faster on System D than on System C. Figure 9 shows the average processor utilization during the execution of the three queries on System C and System D. We see that System C has a higher CPU utilization than System D. A slower processor puts a lower load on the memory system. Hence for the same memory size, processor utilization is more for System C, which has a slower processor compared to System D.

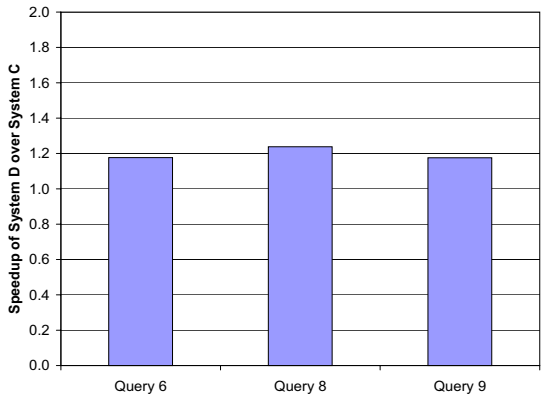

Figure 8: Comparison of Query Execution Time on System C and System D

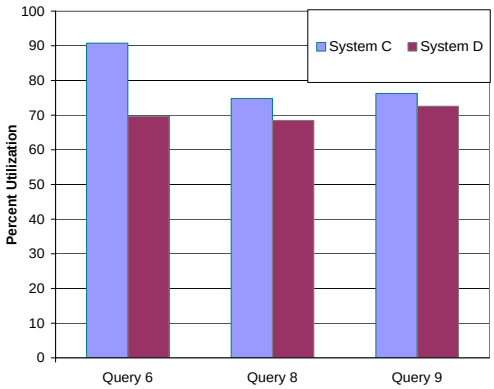

Figure 9: Processor Utilization: System C and System D

6.2 Cache Miss Ratios

Tables 2 and 3 show the L1 and L2 cache miss ratios, respectively, for the three queries on the four systems. Database workloads are known to have high L1 cache miss ratios [4, 9] due to their large memory footprint. Our results show that this also holds true for desktop systems. The average L1 cache miss ratio is over 5 percent for systems A, B and C and around 4 percent for system D. These ratios are on the higher side compared to the average L1 Cache Miss ratio for Spec95 (2.48%) and that for multimedia applications (1.2%) for systems with a similar configuration [16].

We observe that the L2 cache global miss ratios are low on all systems except system C. One of the reasons for the higher cache miss ratio for system C is its smaller cache size (256 KB) as compared to that of the other systems (512 KB). The low cache miss rates for the L2 cache can be attributed to the spatial locality exhibited by all queries. There is spatial locality across the tuples as the relations are sequentially scanned. Further, there is spatial locality within a tuple as it is likely that several of its attributes will be referenced simultaneously. The results lead us to believe that the core working sets of the TPC-H queries fit in the large L2 caches of current desktop systems. The larger block size of the L2 cache captures this spatial locality quite well.

Query Number	System A	System B	System C	System D
6	6.55%	8.50%	8.56%	5.94%
8	5.70%	4.59%	8.32%	2.55%
9	3.59%	3.79%	6.43%	3.37%
Average	**5.28%**	**5.63%**	**7.77%**	**3.95%**

Table 2: L1 Cache Miss Ratios

Query Number	System A	System B	System C	System D
6	0.50%	0.89%	0.76%	0.54%
8	0.34%	0.63%	0.71%	0.42%
9	0.61%	0.27%	0.81%	0.58%
Average	**0.48%**	**0.60%**	**0.76%**	**0.51%**

Table 3: L2 Cache Global Miss Ratios

6.3 Query Analysis

In this section we analyze Query 9 in order to understand its memory referencing behavior. We use the example of Query 9 on System C in this and the following sections whenever it serves as a representative example for all the queries and systems.

Figure 10 shows the graph of paging traffic for Query 9 on System C. The x-axis shows the time in seconds and the y-axis shows the values for the following counters.

- **Page Faults/sec**

  It is the overall rate at which faulted pages are handled by the processor. A page fault occurs when a process requires code or data that is not in its working set (its space in physical memory). This counter includes both hard faults (those that require disk access) and soft faults (where the faulted page is found elsewhere in physical memory). Most processors can handle large numbers of soft faults without any performance degradation. However, hard faults can cause significant delays.

- **Pages Input/sec**

  Pages Input/sec is the number of pages read from disk to resolve hard page faults.

The first join (labeled (1) in the above graph) between the relations, *Nation* and *Supplier*, takes only a few seconds due to their small size. The second join (2) between *Part* and *Line* causes a large number of page faults initially as can be seen from the first peak in the number of pages faults. Most of the initial page faults are hard page faults since neither of the two relations is in memory at that time. The third join (3) is between the temporary relations which were the results of the previous two joins. We see a drop in the number of hard page faults at this point since the temporary relations are already in memory. The fourth and fifth join, join the result of the previous join and the relations *PartSupp* and *Orders*, respectively. This causes an increase in the number of page faults at the beginning of these joins, as seen by the next two peaks (4 and 5) in the graph, since the tuples of the relations are brought from the disk. We see another peak (6) caused by the page faults during the final aggregation.

From this graph, we learn how often the process must look beyond the current working set to find the data it needs. If we compare the number of Page Faults/sec and the number of Pages Input/sec we can determine the proportion of hard page faults to all page faults. In the above graph, the lines for Page Faults/sec and Pages Input/sec intersect when the number of hard page faults equals the total number of page faults. The difference between the two lines indicates soft faults, where the required pages are obtained from elsewhere in the memory, either from the file cache or the working set of other processes. This effect is observed at the beginning of a join and is most noticeable in Figure 10 for join number 4.

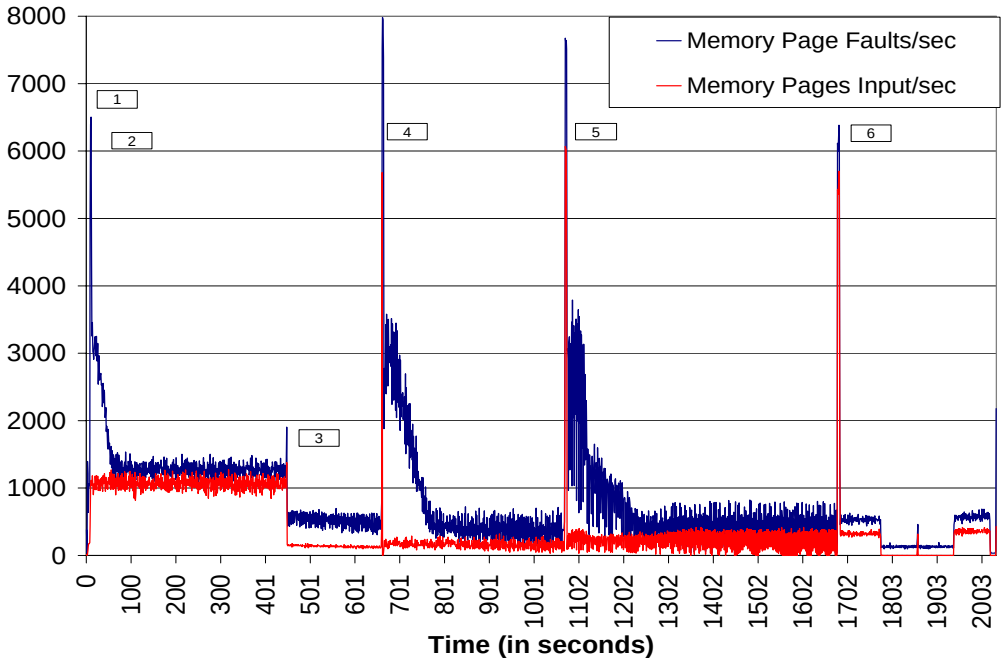

Figure 10: Paging Traffic for Query 9 on System C

6.4 Paging Behavior

Figure 11 shows the average number of pages read per read operation. We use two counters, *Pages Input/sec* and *Page Reads/sec* to obtain the average. It is computed as the ratio of the total number of pages input to the total number of read operations. We have defined the *Pages Input/sec* in the previous subsection. *Page Reads/sec* is defined below.

- **Page Reads/sec**

  Page Reads/sec is the number of times the disk was read to resolve hard page faults. This counter counts number of read operations, without regard to the number of pages retrieved by each operation.

We observe that a large number of pages are read per read operation for all queries. This indicates high sequentiality, which results in high prefetching of data. When we compare across queries we see that a higher number of pages are read per read operation for Query 6 and Query 8. Although Query 9 also displays sequential access to its input relations it cannot maintain a high rate of prefetching as the memory available to the system is lower than that for the other queries. This is because the working set for Query 9 is larger than that for Query 6 and Query 8 (details provided in the next subsection, Section 6.5).

Figures 12 and 13 show the total number of Page Faults/sec and the total number of Pages Input/sec for the three queries for all four systems. It should be noted that we can compare the results of System A only with System B and not with System C and System D because of the difference in the configurations for these systems. Similarly, we can compare the results of System C only with System D. We see from the graphs that the total number of page faults (both hard and soft) increases from left to right as we go from Query 6 to Query 9. This is due to the size and number of the relations involved in the query execution as well as the nature of the query. Query 6 has only one input relation whereas Query 8 and Query 9 have several input relations. From the query definitions, we see that the selection conditions for the tuples of the input relations of Query 9 are less restrictive than those for Query 8. This leads to larger sizes of the

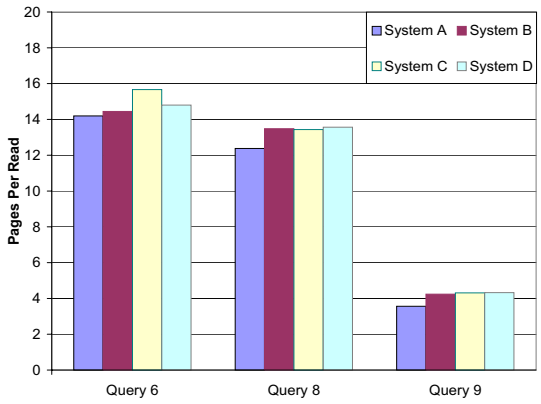

Figure 11: Pages Read Per Read Operation

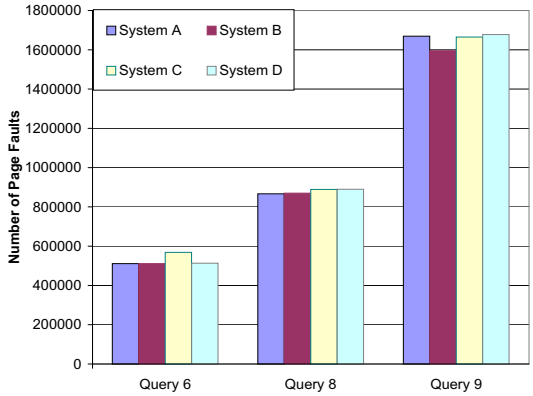

Figure 12: Total Page Faults

temporary relations and thus greater number of page faults and higher execution time for Query 9 as compared to Query 8. In Figure 13, we see that there is little difference in the total number of hard page faults across systems except for Query 9 on System A. We see a higher number of pages input for Query 9 as the working set of Query 9 does not fit in the 256 MB memory for System A.

6.5 Memory Utilization

In this section, we examine the following counters to study the memory utilization for the queries.

- **Working Set**

  Working Set is the current number of bytes in the Working Set of this process. The Working Set is the set of memory pages touched recently by the threads in the process. If free memory in the computer is above a threshold, pages are left in the Working Set of a process even if they are not in use. When free memory falls below a threshold, pages are trimmed from Working Sets. If they are needed they will then be soft-faulted back into the Working Set before they leave main memory.

- **Cache MBytes**

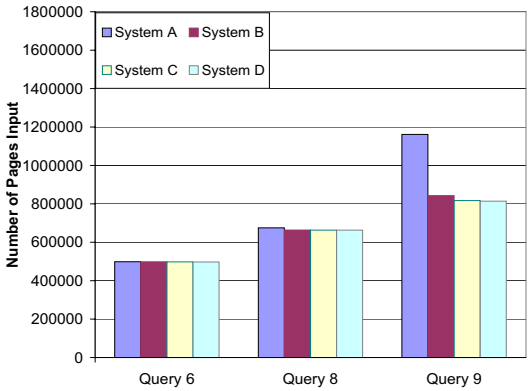

Figure 13: Total Pages Input (Hard Page Faults)

This counter measures the amount of memory utilized by the file cache in MBytes.

- **Available Mbytes**

  Available MBytes is the amount of physical memory available to processes running on the computer, in Megabytes (Bytes/1,048,576). This counter displays the last observed value only; it is not an average.

Figures 14 and 15 show the graphs for the working set of Query 9 on System A and System B. We observe that during query execution the operating system dynamically varies the size of the file cache as required. We see in figure 14 that, for System A, even during the initial stages of query execution the available memory almost drops to 0 MB. Hence the operating system has to frequently free up the memory by writing pages to the disk. This indicates that for System A the working set for Query 9 does not fit in the memory. In figure 15 we observe that, for System B, the available memory is sufficient until the end of query execution. Thus there is no thrashing for Query 9 on System B. From the figures it can be seen that due to thrashing Query 9 takes significantly longer to complete on System A than on System B.

We observe that the working sets for Query 6 and Query 8 are nearly constant throughout query execution. We do not provide their graphs here due to space limitations [1].

6.6 File Cache Performance

We examine the following counters for analyzing the Windows 2000 file cache performance.

- **Copy Reads/sec**

  Copy Reads/sec is the frequency of reads from pages of the file system cache that involve a memory copy of the data from the cache to the application's buffer.

- **Copy Read Hits Percent**

  Copy Read Hits is the percentage of cache copy read requests that hit the cache, that is, they did not require a disk read in order to provide access to the page in the cache.

- **Read Aheads/sec**

  Read Aheads/sec is the frequency of reads from the file system cache in which the Cache detects sequential access to a file. The read aheads permit the data to be transferred in larger blocks than those being requested by the application, reducing the overhead per access.

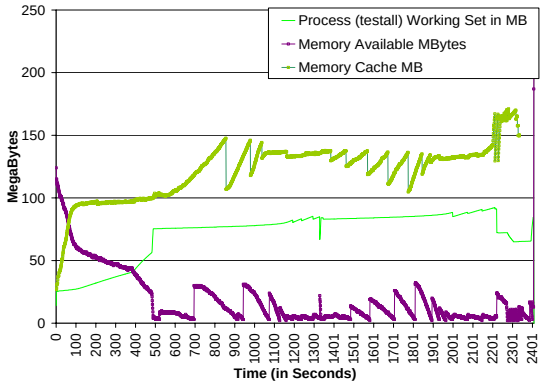

Figure 14: Working Set of Query 9 on System A

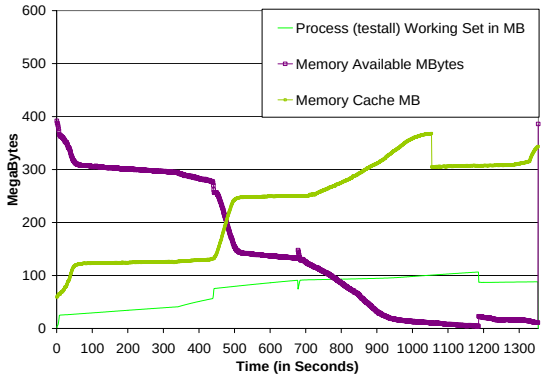

Figure 15: Working Set of Query 9 on System B

Figure 16 shows the plots for the above counters for Query 9 on System C. We see that the copy reads/sec follow the trend of page faults/sec observed in Figure 10. These page faults are generated by the operating system as it prefetches data through the read ahead operation whenever it detects sequential access to a file. We observe that the read aheads/sec, which indicate the amount of prefetched data, vary according to the number of copy reads. This is because the Windows 2000 operating system in combination with the database prefetcher yields a high copy read hit rate. We observe that all queries exhibit a high percentage of copy read hits for all systems. We see that 95% of the time the application finds the data it needs in the file cache.

7 Summary

Our goal was to investigate the performance of database applications, specifically Decision Support Systems, for the desktop environment. We implemented the TPC-H DSS benchmark using the SHORE storage manager on the Windows 2000 platform for this purpose. We list the key observations of our analysis below.

[1] Additional data is provided in [15]

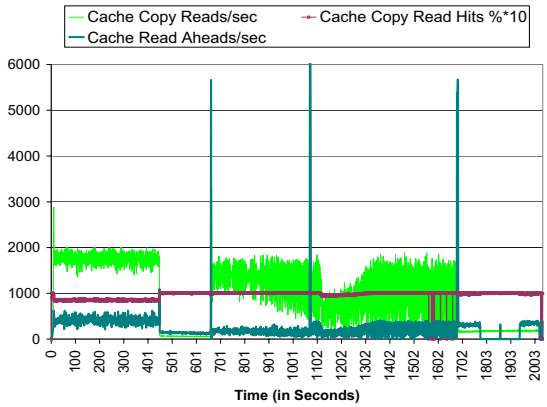

Figure 16: Copy Reads/sec and Copy Read Hit Rate for Query 9 on System C

1. We found that DSS workloads have high L1 cache miss rates. In contrast, we found that the L2 cache miss rates are low. The results lead us to believe that the core working sets of the TPC-H queries fit in the large L2 caches of current desktop systems.

2. We observed that on an average, a large number of pages are read per read operation. This indicates that there is a high amount of sequentiality in the database accesses. It also indicates that the Windows 2000 operating system along with the database prefetcher detects this sequentiality and initiates high amount of prefetching.

3. We found that the performance of the file cache is very good, with copy read hit rates being more than 95 percent for all systems. This is surprising since the Windows 2000 file system cache is not optimized for database applications. We attribute the high file cache hit ratio to its self-tuning prefetching mechanism that detects the sequentiality of database accesses.

4. We also found that the performance of the queries and the scalability of the database depend on the nature of the queries as well as the size and contents of the database.

5. We believe that increasing memory beyond the point where the working set of a query fits in the memory will yield diminishing results. To further improve the performance we suggest increasing the amount of data prefetching. This can be done either through the operating system or through the application.

6. The Microsoft TechNet guide [7] specifies a framework for identifying system bottlenecks. To find whether there are any bottlenecks for DSS workloads we measured counters for various system resources such as processor, memory and disk. For the counters that we measured, the counter values were well below the limits specified by the TechNet guide indicating that there are no bottlenecks.

Based on these observations, we believe that the desktop environment can support small to medium scale database applications.

References

[1] A. Ailamaki, D. DeWitt, and M. Hill. Data page layouts for relational databases on deep memory hierarchies. *The VLDB Journal*, 11(3), 2002.

[2] A. Allamaki, D. DeWitt, M. Hill, and D. Wood. DBMSs on a modern processor: Where does time go? In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*, pages 266–277, September 1999.

[3] M. Annavaram, J. Patel, and E. Davidson. Call graph prefetching for database applications. In *HPCA*, pages 281–, 2001.

[4] L. Barroso, K. Gharachorloo, and F. Bugnion. Memory system characterization of commercial workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA-98)*, volume 26,3 of *ACM Computer Architecture News*, pages 3–14, June 27–July 1 1998.

[5] M. Carey, D. Dewitt, M. Franklin, N. Hall, M. McAuliffe, J. Naughton, D. Schuh, M. Solomon, C. Tan, O. Tsatalos, S. White, and M. Zwilling. Shoring up persistent applications. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):383–394, June 1994.

[6] Intel Corporation. Vtune performance analyzer. http://www.intel.com/software/products/vtune.

[7] Microsoft Corporation. Microsoft technet. http://www.microsoft.com/technet/default.asp.

[8] Transaction Processing Performance Council. TPC benchmark H standard specification. Technical report, Transaction Processing Performance Council, june 1999.

[9] Z. Cvetanovic and D. Donaldson. Alphaserver 4100 performance characterization. Technical report, Digital Equipment Corporation, November 4, 1997.

[10] D. DeWitt, N. Kabra, J. Luo, J. Patel, and J. Yu. Client-Server Paradise. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 558–569, Santiago, Chile, 1994.

[11] J. Patel et al. Building a scalable geospatial database system: Technology, implementation, and evaluation. In *SIGMOD*, 1997.

[12] Y. Zhang et al. Characterizing TPC-H on a clustered database engine from the os perspective. In *HPCA*, 2002.

[13] Benchmark Insider. Cpumark. http://www.etestinglabs.com/bi/cont1999/1999print/scoring.asp, April 1999.

[14] M. Karlsson, F. Dahlgren, and P. Stenstrom. An analytical model of the working-set sizes in decision-support systems. In *Measurement and Modeling of Computer Systems*, pages 275–285, 2000.

[15] S. Kavalanekar and Y. Hu. Performance analysis of decision support workloads for the desktop environment. Technical report, University of Cincinnati, http://www.ececs.uc.edu/õscar/papers/dss.html, July 2003.

[16] S. Sohoni, Z. Xu, R. Min, and Y. Hu. A study of memory system performance of multimedia applications. In *Proceedings of the ACM Joint International Conference on Measurement & Modeling of Computer Systems (SIGMETRICS 2001)*, pages 206–215, Cambridge, Massachusetts, June 16– 21 2001.

[17] P. Trancoso, J. Larriba-Pey, Z. Zhang, and J. Torrellas. The memory performance of DSS commercial workloads in shared-memory multiprocessors. In *Proceedings of the Third International Symposium on High Performance Computer Architecture (HPCA '97)*, pages 250–260, February 1997.