Britton Lee, Inc.

111 N. SEPULVEDA BLVD.
MANHATTAN BEACH, CA 90266
(213) 318-7009

DON SMITH
DIRECTOR
WESTERN REGIONAL SALES

May 6, 1986

System Development Corp.
2500 Colorado Ave. M.S. 91-10
Santa Monica, CA 90406

Enclosed are various benchmarks detailing transaction rates. Please note they were performed with a 6 MHz DBP and SMD Disk Controllers. I've included a benchmark (#11) which shows the relative performance of the 10 MHz and 6 MHz processor. This will give you an idea of the performance improvement. Unfortunately, there are no published SMD vs. SMDE benchmarks as it was just released from beta. I would like to mention that the SMDE controller supports a transfer rate of 3.0 Mbytes/sec., which more than doubles the current SMD Controller's 1.2 Mbytes/sec. maximum rate.

Please let me know if I can be of further assistance.

Sincerely,

C. Williams
Sales System Engineer

CW:ns
Encls.

IDM Performance Test 1:
Simple Retrieves Varying the Number of Users

Michael Tossy

Purpose

This performance test was designed to characterize the effect on IDM performance as the number of users grew.

Overall summary

We demonstrated that the IDM has greater "throughput" as the number of users is increased. The IDM with 24 users processed querys at 2.73 times the single user rate.

Background

As users are added to a computer system the "throughput" (the rate at which the system does work) changes.

Normally two effects -- one negative, one positive -- are seen when adding users to a system. In general additional users increase the system overhead, swapping is an example of this negative effect. But additional users also allow overlapping processing. One user executing (using the CPU) while another is doing I/O (using the disk controller) is an example of this positive effect. Up to a point, more efficent resource utilization more than compensates for the additional overhead.

Some systems show a negative throughput curve when they become overloaded. For example, most timesharing systems "thrash" when overloaded.

Benchmark Background

The benchmark simulates a series of production type queries by multiple users. All queries were RETRIEVES with a simple restriction on one attribute. The attribute had either a clustered or nonclustered index. A single relation of 150,000 tuples was used. The relation had nine attributes, and the average tuple was 100 bytes wide. About two thirds (2/3) of the retrieves were on the nonclustered attribute, the other one third (1/3) were on the attribute with the clustered index.

The benchmark was done under UNIX. Scripts of IDL Queries were prepared. Each script had ten (10) retrieves. Multiple users were simulated by spawning several IDL processes each reading from a different script. A single master process was used to spawn the individual user

processes, the times were obtained by timing the master process. This procedure restricted us to simulating a maximum of 24 users.

All relations on the IDM were created with logging. However, since no updates were done, this did not affect performance.

The Conditions

The host computer was a VAX 11/750 running UNIX BSD 4.1. The IDM was a 500 with a Database Accelerator and 1.5 Megabytes of memory. The IDM had a single 160 Megabyte disk drive. The IDM was running release 24 of the IDM code (release 1 of the WCS). The IDM and host were connected by a RS 232 serial connection running at 9600 baud.

Both host and IDM were idle except for the benchmark.

Data collection

The Ad Hoc queries were run using a "master" shell script on UNIX (4.1 BSD). This "master" shell script spawned all the user processes and then waited until they all completed. The benchmark timings were of the "master" shell script obtained by the "time" command. These values were then stored on the IDM. The IDM was used to display and calculate the performance numbers from the collected data.

Timings

This table shows the number of concurrent users, the time to run all the queries, and the system throughput.

The attributes are:

users	the number of users
time	the time to run the queries
per_user	users divided by time
thruput	(users * 12.1) / time

users	time	per_user	thruput
1	12.1	12.	1.
2	14.330	7.1760	1.6880
4	22.8	5.70	2.1230
8	37.6	4.70	2.5740
14	63.0	4.50	2.6890
16	72.8	4.55	2.6590
18	83.170	4.62	2.1690
24	106.20	4.4250	2.7340

Timing summary

As you can see, to run these queries as a single user takes about 12 seconds. As we add users, and queries, the times, as you would expect, increase.

Look at the first record. The attribute "per_user" shows one user took 12.1 seconds to run. Now look at the second record. A value of 12.1 in this field would mean that two users took twice as long as a single user. A value greater than 12.1 means that the system is running slower with multiple users. A value less than 12.1 means that some effective overlap of resources is going on. The real value is 7.17, that means considerable overlap is occurring.

This same effect is shown in the "thruput" attribute. Here we show the amount of work done by the IDM per unit time, normalized so that the single user case is one. You can see that the "thruput" attribute increases, up to 2.73 for 24 users. This means that, with 24 users, throughput of the IDM is 1.73 times greater than with 1 user.

Overhead

The simulation process had some overhead. We tried to estimate the overhead by running the master script modified to spawn zero user processes. It consistently took two (2) seconds to run thebenchmark with zero (0) users.

Extensions to the Benchmark

There are several additional tests that could be done to extend the usefulness of this benchmark:

Rerunning the tests without an accelerator.
Rerunning the tests on an IDM 200.
Using a greater mix of queries. Including some replaces, and appends, or some retrieves that do not use any indices.
Simulation of more users.
Use of multiple hosts.

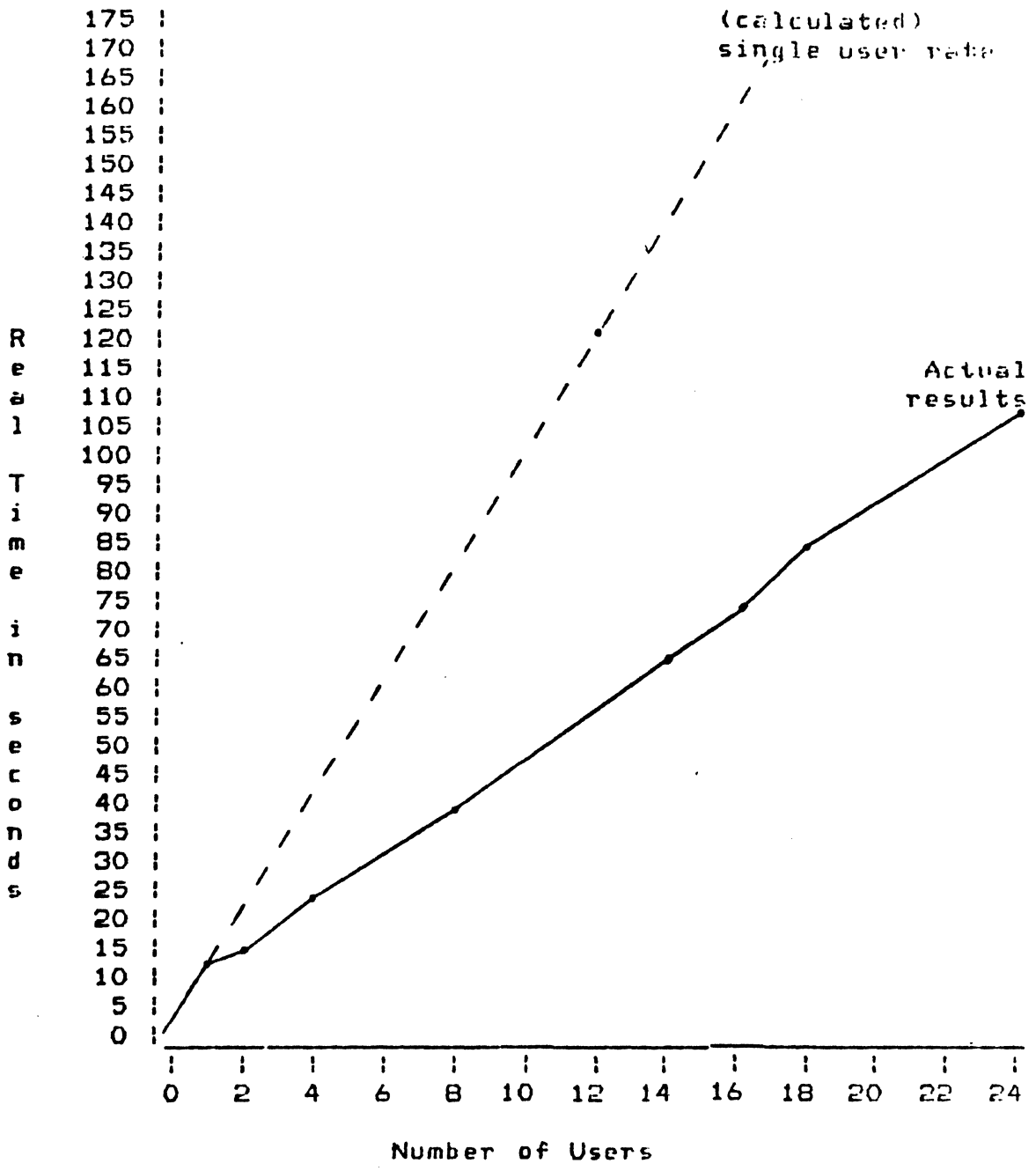
Summary

We demonstrated that the IDM has greater "throughput" as the number of users is increased. The IDM with 24 users processed querys at 2.73 times the single user rate.

Appendix A: The Data

users	time
0	2.
0	2.
0	2.
0	2.
0	2.
1	13.0
1	11.0
1	13.0
1	12.0
1	13.0
1	12.0
1	13.0
1	11.0
1	12.0
1	11.0
2	14.0
2	15.0
2	14.0
2	13.0
2	14.0
2	16.0
4	22.0
4	21.0
4	24.0
4	24.0
4	23.0
8	42.0
8	37.0
8	36.0
8	36.0
8	37.0
14	61.0
14	61.0
14	62.0
14	68.0
16	67.0
16	70.0
16	79.0
16	75.0
16	73.0
18	79.0
18	83.0
18	87.0
18	80.0
18	84.0
18	86.0
24	100.0
24	97.0
24	110.0
24	112.0
24	112.0

Appendix B: graphing the results



IDM Performance Test 2:
Simple Retrieves Varyingthe Number of Tuples

Michael Tossy

Purpose

This performance test characterizes the performance of the IDM doing retrieves on relations of different sizes.

Overall summary

We proved that, for queries that do not use an index, the IDM retrieval time is linearly dependent on the number of tuples. Queries that use an index do not show this dependency, those retrieval times seem to be nearly constant for the range examined. Clustered indices are about 10% faster than nonclustered indices.

We also demonstrated that the Database Accelerator always speeds retrieval times. For retrievals using an index, the Database Accelerator improves query times by about 140%.

Background

The time to run a query on the IDM is known to be dependent on many factors: complexity of where clause(s), number of tuples examined, number of tuples that qualify, number of attributes in relation, and number of relations, are just some of them. Two major factors are the access method (type of index used) and the presence, or absence, of a Database Accelerator (DAC).

There are three methods used, by the IDM, to access relations. The simplest, and slowest, method is a "scan" of the relation. A "scan" simply reads all tuples and checks if they meet the qualification. This method is normally used only when other methods are not available. The other two methods use a B* tree index to give direct access to the desired tuple(s). A "clustered index" sorts the relation into an order based on a specified attribute(s), this provides quick access when the query includes a restriction on that attribute(s). A "nonclustered index" is similar, but does not sort the relation, therefore it must build a more complex index. We would expect the clustered index to be faster than the nonclustered index.

Benchmark Background

This benchmark is designed look a query speed as a factor of relation size. We examined relations of eight different sizes (100, 1000, 5000, 10000, 25000, 50000, 100000, and 150000 tuples). We examined these relations using all six combinations of access method (no index, clustered index, and nonclustered index) and Database Accelerator (present and absent). All other factors were held constant.

The benchmark simulates a series of queries by a single user. All queries were RETRIEVES with a simple restriction on one attribute. That attribute had either a clustered or nonclustered index.

The relations had nine attributes, and the average tuple was 100 bytes wide. All relations on the IDM were created with logging. However, since no updates were done, this did not affect performance.

The Conditions

The host computer was a VAX 11/750 running UNIX BSD 4.1. The IDM was a 500 with 1.5 Megabytes of memory. The IDM had a single 160 Megabyte disk drive. The IDM was running release 24 of the IDM code (release 1 of the WCS for runs with an Accelerator). The IDM and host were connected by a RS 232 serial connection running at 9600 baud.

Both host and IDM were idle except for the benchmark.

Data collection

IDM "set 5" times were used. These are times, returned by the IDM, that indicate the real time inside the IDM from beginning to end of the query. These times include all processing within the IDM and all data transfer between the Host and the IDM, they do not include "parsing" time on the Host.

These times were stored on the IDM, and the IDM was used to display and calculate the performance numbers from the collected data.

Timings by relation size

(The data displayed in this section has been graphed. The graphs are in Appendix A.)There are six tables, one for each combination of DAC usage (yes or no) and index type (clustered, nonclustered, and no index). Each table shows the number of tuples and the average time to run a simple retrieve in that configuration.

The attributes are:
number of tuples (in the relation)
access speed (in milliseconds)

number of tuples	access speed
100	128.75
1000	137.25
5000	170.50
10000	191.25
25000	178.75
50000	183.
100000	195.75
150000	195.50

Table 1
Access speed: DAC, clustered index

number of tuples	access speed
100	145.75
1000	150.
5000	174.75
10000	191.50
25000	178.75
50000	212.
100000	233.
150000	195.50

Table 2
Access speed: DAC, nonclustered index

number of tuples	access speed
100	158.
1000	250.
5000	8203.75
10000	16462.25
25000	40716.50
50000	85541.
100000	176666.25
150000	268270.50

Table 3
Access speed: DAC, no index

number of tuples	access speed
100	308.
1000	266.
5000	345.50
10000	466.50
25000	491.50
50000	558.
100000	583.
150000	349.75

Table 4
Access speed: no DAC, clustered index.

number of tuples	access speed
100	345.75
1000	316.25
5000	391.50
10000	503.75
25000	545.50
50000	637.25
100000	653.50
150000	437.25

Table 5
Access speed: no DAC, nonclustered index

number of tuples	access speed
100	478.75
1000	983.
5000	10507.75
10000	21157.75
25000	51966.25
50000	106262.25
100000	218774.75
150000	318745.5

Table 6
Access speed: no DAC, no index.

Timing discussion

Looking at table 1 (clustered index and DAC) we can see that the access time is fairly constant. The number of tuples varies by three orders of magnitude (1500 times more tuples), yet the difference between the fastest and slowest query is only 52% (1.52 times as long). The first two cases (100 and 1000 tuples) had only a single level index, while the others had a two level index index structure, that may explain the large jump difference in query times between those two groups.

Table 4 (clustered index, no DAC) shows a similarly consistent retrieval time, varying by only 119% (2.19 times as long). The DAC times are consistently faster (see later section). The effect of the additional index level is not as clear in this data. The 150,000 tuple case is markedly faster than expected; the cause isn't known.

The nonclustered index data (Tables 2 and 5) show the same consistent times as the clustered indices. Query times for table 2 vary by only 59% (1.59 times as fast). Query times for table 5 vary by 107% (2.07 times as fast). Again, the DAC always helps (see later section) and the clustered index is almost always faster than the nonclustered index (see later).

Now look at table 3 (DAC, no index). Starting with the 5000 tuple case we get times that increase linearly with the number of tuples, just as we would expect. (Below 5000 tuples this effect is clouded by other factors, such as query analysis, and disk cacheing).

Table 6 (No DAC, no index) shows the same effect as table 3; the query times go up linearly with the number of tuples. The effect of the DAC is still noticeable, although not as significant. These querys require a complete scan of the relation and are therefore I/O bound.

Timing summary

Access times are do not vary significantly for indexed queries. For unindexed queries the time is linearly proportional to the number of tuples.

This information is particularly useful to users who wish to prototype a large system. Access times can be predicted by prototyping a 10th, 100th, or even 1000th scale model of the production database.

Comment

There is no direct way to discover how many levels there are in an IDM index. By calculation, I would have expected three nonclustered index levels for the 25,000, 50,000, 100,000, and 150,000 tuple relations, two levels for 1,000, 5,000, and 10,000 tuples, and one level for 100 tuples. The query times do not seem to support this hypothesis.

Effect of Accelerator on times

There are three tables, one for each access type (clustered index, nonclustered index, no index). Each table shows the average effect of the DAC, and the effect of the DAC by relation size.

The attributes are:
number of tuples (in the relation)
Time without DAC divided by time with DAC

Overall improvement for this access method:

number of tuples	no DAC time / DAC time
100	2.1905
1000	1.8652
5000	2.0022
10000	2.4637
25000	3.0245
50000	3.2679
100000	2.6051
150000	1.7819

average improvement
2.4043

Table 7: Increase in speed using DAC, access via clustered index.

number of tuples	no DAC time / DAC time
100	2.3861
1000	2.0626
5000	2.1847
10000	2.5404
25000	2.9699
50000	2.9898
100000	2.7663
150000	2.1117

average improvement
2.5258

Table 8: Increase in speed using DAC, access via nonclustered index.

number of tuples	no DAC time / DAC time
100	2.9026
1000	2.2251
5000	1.2727
10000	1.2652
25000	1.2676
50000	1.2290
100000	1.2178
150000	1.1743

average improvement
1.2067

Table 9: Increase in speed using DAC, access via scan (no index).

DAC effect discussion

Tables 7 and 8 represent production type queries, and both tables 7 and 8 show a consistent boost in performance with the DAC. Table 9 is interesting since it shows a consistently declining (as the relation size increases) boost by the DAC. This was expected, because these queries are I/O bound and tend to become more so as the size of the relation increases.

DAC effect summary

The DAC consistently helps all queries. The CPU bound queries are helped more than the I/O bound queries. Non-indexed queries were helped about 20% by the DAC; index queries were helped between 140% and 150%.

Effect of Index on times

This section shows the difference in query times between clustered and nonclustered index access. The sub-tables show the difference by relation size and the average of all cases.

The attributes in the left sub-table are:
number of tuples (in the relation)
presence of Database Accelerator (DAC)
nonclustered index time divided by clustered index time

number of tuples	DAC	improvement in speed
100	N	1.122560
1000	N	1.188910
5000	N	1.133140
10000	N	1.079850
25000	N	1.109870
50000	N	1.142030
100000	N	1.120930
150000	N	1.250180
100	Y	1.132040
1000	Y	1.092900
5000	Y	1.024930
10000	Y	1.001310
25000	Y	1.000000
50000	Y	1.158470
100000	Y	1.190290
150000	Y	1.000000

average increase
1.118550

average increase with DAC
1.072790

average increase without DAC
1.137310

Table 10
Speed advantage of clustered
index over nonclustered index

Index effect summary

In general the clustered index seems to have about a 10% speed advantage over the nonclustered index. The presence of the DAC seems to moderate the difference.

Overall summary

We demonstrated that, for queries that do not use an index, the IDM retrieval time is linearly dependent on the number of tuples. Queries that use an index do not demonstrate this dependency, rather the retrieval times seem to be nearly constant for the range examined. Clustered indices are faster than nonclustered indices, but not substantially so.

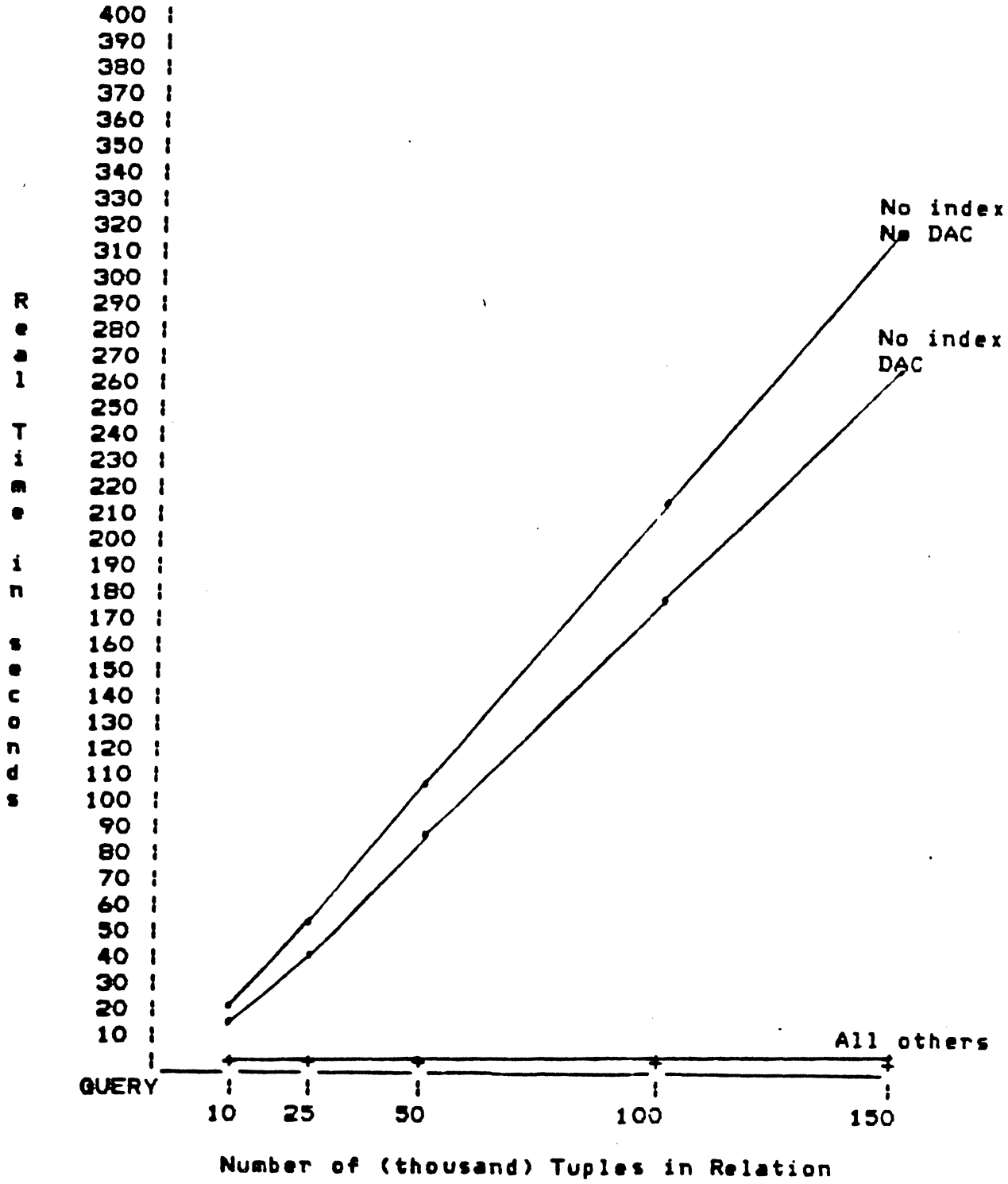
We also demonstrated that the Database Accelerator always speeds retrieval times; in production type queries the DAC more than doubled the IDM performance.

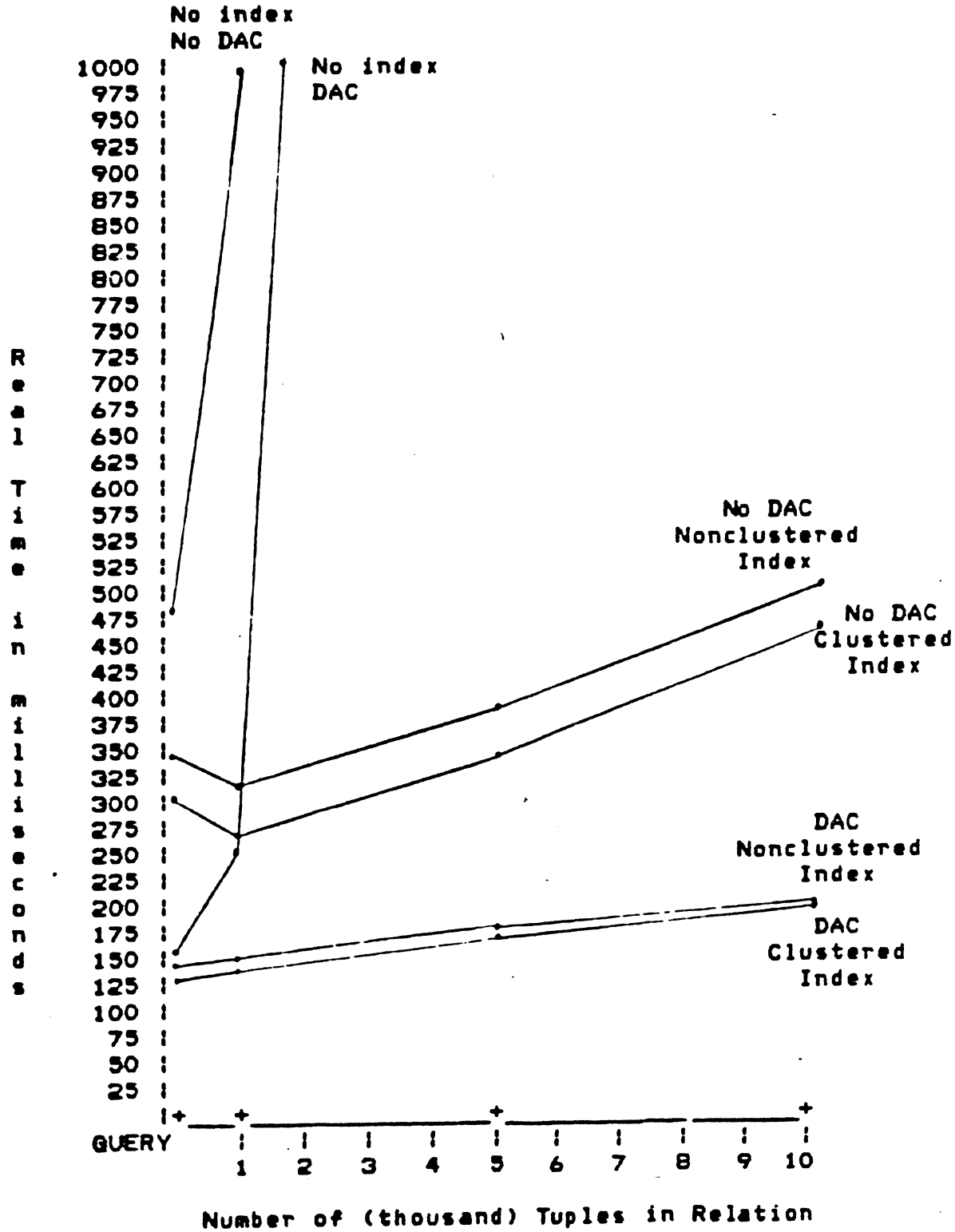
Extensions to the Benchmark

There are several additional tests that could be done to extend the usefulness of this benchmark:

Increasing the number of tuples.
Rerunning the tests on an IDM 500/0.
Repeating the test with queries that require joins.
Investigate why the 150,000 tuple relation runs faster than the 100,000 tuple relation.
Repeat test using stored commands.

Appendix A: graphing the results




No index
No DAC
No index
DAC
No DAC
Nonclustered
Index
No DAC
Clustered
Index
DAC
Nonclustered
Index
DAC
Clustered
Index
Real Time in milliseconds
1000
975
950
925
900
875
850
825
800
775
750
725
700
675
650
625
600
575
525
525
500
475
450
425
400
375
350
325
300
275
250
225
200
175
150
125
100
75
50
25
QUERY
1 2 3 4 5 6 7 8 9 10
Number of (thousand) Tuples in Relation

Date: 11 April 1985
Section: 3 (IDM performance)
Subsection: 3 (The IDM only)
Article: 7 (#5 of The Britton Lee IDM Performance Study Series)

Title: IDM Performance Test #5: DAC Performance
Author: Ed Simon

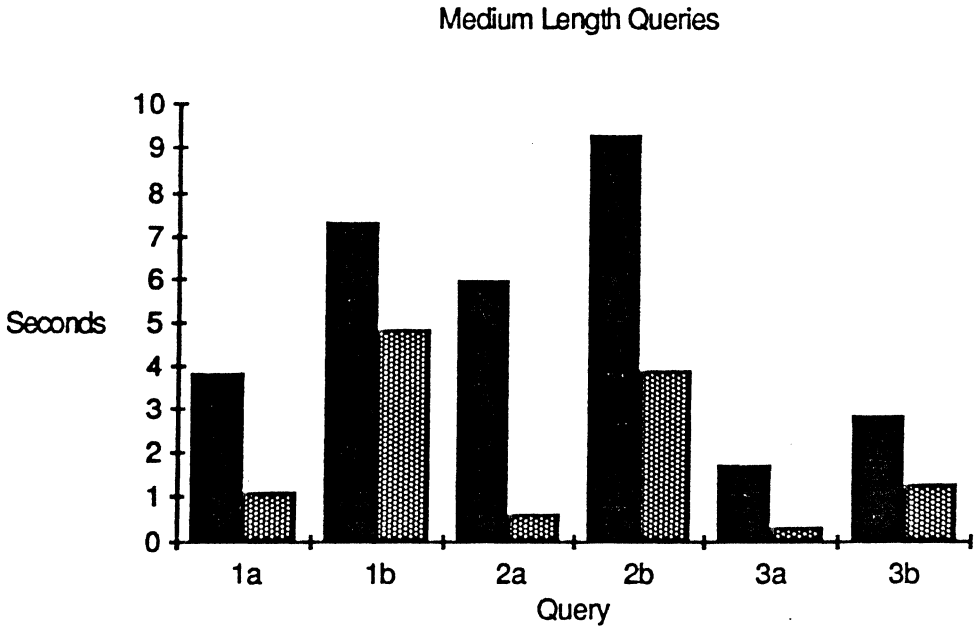
Synopsis: Part of a series of papers maintained by the Britton Lee Sales Support group and designed to point out various aspects of the IDM's performance. All of the previous papers study the effect of the DAC on the performance of a specific simple command but this paper looks at the effects of the DAC in a wide variety of operations. It also separates the effects of disk I/O time from CPU performance in evaluating the DAC improvements.

IDM Performance Test 5:
DAC Performance

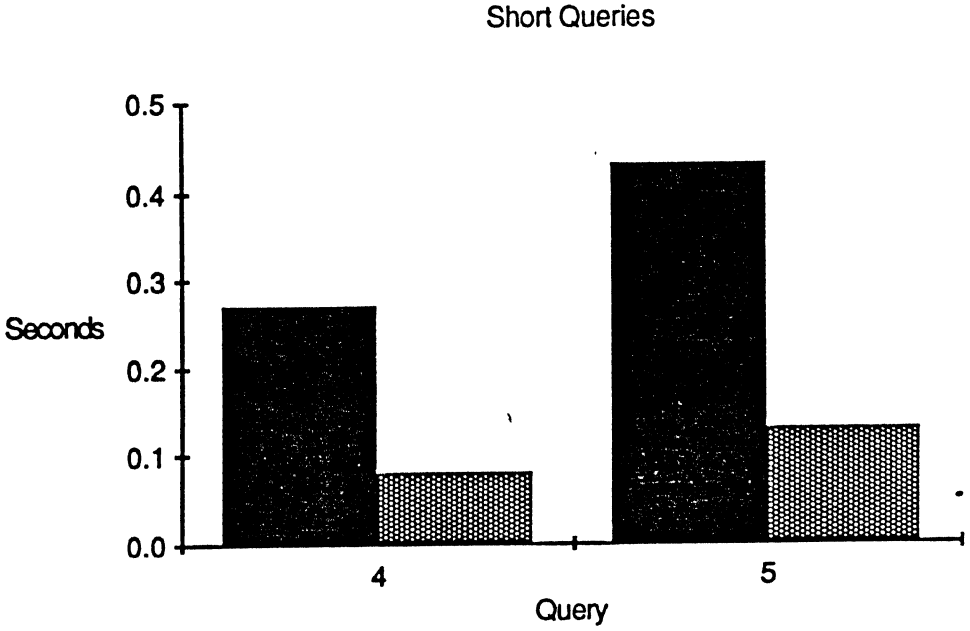
Ed Simon

Summary

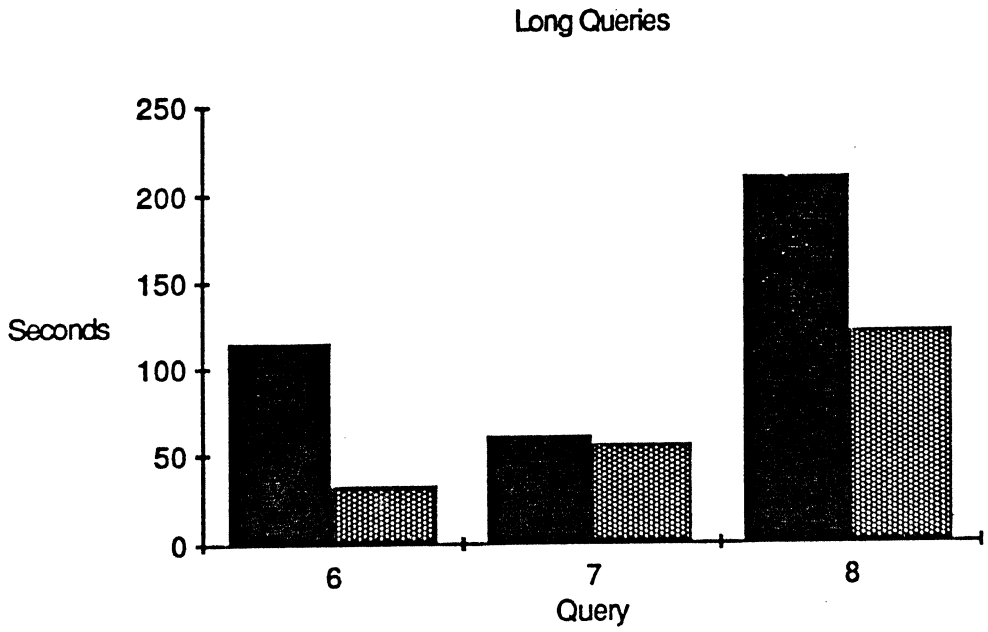
The major difference between the IDM 500/1™ and the IDM 500/2™ is that the IDM 500/2 includes the Database Accelerator™ (DAC), an optional high performance "search engine". As can be seen below, the DAC significantly improves the IDM's performance on some queries.



(Time without DAC shown on left, time with DAC shown on right)



(Time without DAC shown on left, time with DAC shown on right)



(Time without DAC shown on left, time with DAC shown on right)

Two general statements can also be made about the results of this study:

- A query that is CPU bound will show larger relative performance improvement than a query that is disk I/O limited.

- The improvement in performance can be estimated by looking at the DAC time, SET 44, returned by the IDM.

Note

This report replaces the original report on DAC performance dated March 1983. In this report, some of the experiments performed in the earlier report are repeated with the current software revisions (IDM release 31, UNIX release 3 IDL).

Background

The Database Accelerator (DAC) is a high performance (8 MIPS), single board processor which plugs into any free slot of an IDM 500. The major difference between the IDM 500/1™ and the IDM 500/2™ is that the IDM 500/2 includes the Database Accelerator™ (DAC). The DAC offloads certain functions from the microprocessor based Database Processor (DBP). The major improvement in IDM performance contributed by the DAC occurs largely from exploiting the "90 - 10 rule" (90% of the time is spent in 10% of the code). By placing that 10% in a very fast processor (about 10 times faster than the main processor) Britton Lee has achieved significant performance improvements. (In a relational system much time is spent looking for patterns in records, and that is the DAC's primary function.)

When the DAC is not present the DAC functions are simulated by the slower main processor (DBP). The "DAC" attribute in the IDM monitor relation and the SET 44 time both indicate how much time is spent by the DAC (if present) or by the DBP simulating DAC time (if the DAC absent).

Introduction

From the descriptions in the background section you can guess that the impact of the DAC depends on the query being run. Furthermore it should be possible to estimate the impact of adding a DAC to an IDM without one, by looking at either the SET 44 times or DAC attribute in the monitor relation. A rule of thumb for estimating the benefits of adding the DAC assistance to an existing application is to look at the value returned by the SET 44 option on the query. Assume that 90% of this time would be saved with a DAC. This study will be used to verify that assertion.

This report is meant to give easily reproduced results on a simple set of queries against a small, simple relation. More complete studies that observe relative performance with and without the DAC have been published. The most informative of these is the pair of studies by Boral and DeWitt [BORA84] and Hawthorn [HAWT85].

All experiments in this test use a relation with 24001 tuples on about 170 pages. Tne relation is:

```
create dictionary (word=c30, length=i2) go
create clustered index on dictionary (word) go
```

All of the studies in this report were done using the performance reporting options on the IDM. This allows us to control the query relative to disk performance as well as observing the effect of wait queues on results. The times with the DAC were performed on an IDM 500/2 (with 2 Megabytes of memory and a DAC). The times without a DAC were performed on the same IDM 500/2 with the DAC software disabled and thus simulating a 500/1 with 2 Megabytes of memory. None of the studies involve returning large amounts of data to the host, so that host I/O wait times as well as disk access times could be controlled.

Simple Retrieves

These examples show processing time saved by using a DAC on simple indexed and unindexed retrievals. For each experiment in this section, the times were measured with the data resident on disk and with the data already in cache memory. The following command performs a minimal amount of work on each tuple in a sequential scan:

Example 1

```
range of d is dictionary
retrieve (x=count(d.word)) go
```

Condition	In memory	On disk	0.9 x SET 44
Without DAC:	3.8 sec	7.3 sec	2.8 sec
With DAC:	1.1 sec	4.8 sec	
Difference:	2.7 sec	2.5 sec	
Factor:	3.5 times	1.5 times	

Notice that the difference in processing time for this 24,000 tuple scan between the DAC and No DAC cases is independent of the disk access time. The Factor shows how many times faster the process proceeded with the DAC. For comparison, 90% of the SET 44 value returned in the No DAC case is also given. In this case the actual time saved was about 2.6 seconds; the SET 44 rule predicts 2.8 seconds.

In the next example, a restriction is placed on the unindexed length attribute:

Example 2

retrieve (x=count(d.word where d.length >= 13)) go

Condition	In memory	On disk	0.9 x SET 44
Without DAC:	5.9 sec	9.3 sec	5.1 sec
With DAC:	0.6 sec	3.9 sec	
Difference:	5.3 sec	5.4 sec	
Factor:	9.5 times	2.4 times	

Here the effect of the DAC is more dramatic; the SET 44 rule even underestimates the savings. The additional restriction forces both fields of each tuple to be referenced, nearly doubling the processing time in memory. Note that the time to read the relation from the disk is essentially fixed.

In the next example we place a restriction on the indexed attribute that allows the query to reference fewer pages:

Example 3

retrieve (x=count(d.word
where d.word = "s" and d.length >= 13)) go*

Condition	In memory	On disk	0.9 x SET 44
Without DAC:	1.7 sec	2.8 sec	1.6 sec
With DAC:	0.3 sec	1.3 sec	
Difference:	1.4 sec	1.5 sec	
Factor:	6.3 times	2.2 times	

Again the SET 44 value slightly underestimates the savings.

Simple Updates

A single update does not require scanning much of the query, so it tends to run relatively quickly. The time to append a tuple does not depend on the size of the relation. In this example we simply add a word to the relation. For the balance of the examples, the processing time from disk will not be included:

Example 4

append to dictionary (word="IDM", length =3) go

Condition	Total time	0.9 x SET 44
Without DAC:	.27 sec	.18 sec
With DAC:	.08 sec	
Difference:	.19 sec	
Factor:	3.4 times	

The next example does a replace on a single tuple, that can be located with one traversal of the index:

Example 5

replace d (word="DataBaseManagementSystem", length=24)
where d.word = "databasemanagementsystem"
and d.length = 10 go

Condition	Total time	0.9 x SET 44
Without DAC:	.43 sec	.24 sec
With DAC:	.13 sec	
Difference:	.30 sec	
Factor:	3.3 times	

In the next example, a global Replace is performed, where all words beginning with "s" are replaced by their current value (this doesn't change the relation, but requires the same processing time). In this case, only part of the relation is scanned, but this processes over 2200 updates for this relation.

Example 6

replace d (word= d.word) where d.word = "s" go*

Condition	Total time	0.9 x SET 44
Without DAC:	114. sec	81. sec
With DAC:	32. sec	
Difference:	82. sec	
Factor:	3.6 times	

Conclusion

We have seen several reproducible and useful results from this study:

- The Factor of improvement may not be the best measure of DAC enhancement.

- The formula that specifies that 90% of the SET 44 value returned by the IDM can be saved by installing a DAC is very accurate for most common queries.

- The time saved by DAC processing is all CPU time. Disk access time has no effect on the saving.

Bibliography

[BORA84] Boral, H., De Witt, D., "A Methodology for Database System Performance Evaluation", Proceedings, SIGMOD, Boston, 1984.

[HAWT85] Hawthorn, P., "Variations on a Benchmark", Britton Lee Inc., 1985.

Date: 28 July1985
Section: 3 (IDM performance)
Subsection: 3 (The IDM only)
Article: 10 (#8 of The Britton Lee IDM Performance Study Series)

Title: IDM Performance Test 10: Mirrored Disk Performance
Author: Ed Simon

Synopsis: The mirrored disk option was introduced to improve data reliability and availability by addressing the issue of hardware failure at the weakest point in the system, the disk drives. To extend the protection from hardware failure BLI recommended that mirrors be placed on separate controllers.

Initial performance expectations for the option were that:

- Disk read operations would be faster.
- Write operations would be slower.
- The typical application would run about the same speed with or without mirrored disks provided disk pairs were connected to separate controllers.

In general, these expectations are borne out with the exception that multiuser disk reads are so much faster with mirrored disks (on a single as well as separate controllers) that mirroring should be considered a significant option for the performance improvement of some applications.

IDM Performance Test 8:
Mirrored Disk Performance

Ed Simon
June 1985

Summary

The performance of Mirrored disks on the Britton-Lee IDM 500 was measured for a variety of user processes and compared to the performance for the same processes on non-mirrored disks. In general, update performance for mirrors with a single disk controller was somewhat degraded (5% to 15%), while multi-user disk reads were observed to run up to 260% faster on Mirrored disks.

Adding a second disk controller improved performance considerably. Mirrored updates ranged from 6% faster to 10% slower. Reads were up to 340% faster for Mirrored disks in multi-user situations.

Introduction

The Mirrored disk feature of the IDM is attractive for reasons of data security. The Mirrored disk feature permits one disk device to be a physical mirror of a second device, thus protecting the physical database from losses due to single disk failures. In addition, some hard disk errors are automatically repaired by the block being remapped and copied from the unaffected mirror. When combined with the other data security and integrity features of the IDM, the Mirrored disk option provides a capable backup and integrity system for user data, while optimizing on-line availabilty.

In general, users expect to pay some performance cost for data security, so to investigate this implication, we measured the time necessary to complete a variety of tasks on the IDM. We expected updates to run somewhat slower, due to the fact that two disks had to be written for each update. We also expected data retrieval to run somewhat faster due to the availability of a second drive.

The algorithm used for reading disk data with Mirrored drives permits each block request to be filled by the drive closest to that disk block. To keep one drive from getting all of the exercise on reads, the algorithm is reversed every so often. For writes, the data is written to bothdevices simultaneously, so there may be an extra cost of waiting until both disk operations have completed. This cost was expected to be higher for a system with a single disk controller.

Configuration

The benchmark was performed on an IDM 500/2 with a database accelerator (DAC), 2 Megabytes of memory (450 disk cache pages), serial connection to a VAX 785 running Ultrix and parallel connection to a VAX 11/750 running VMS. The IDM had 3 160-megabyte disk drives initially on a single controller. The IDM software was release 32. Drive 1, the Fujitsu system drive, was not Mirrored while drives 2 and 3, CDC drives, were Mirrored. At one point in the experiment, one of the CDC drives was intentionally downed to see that the IDM continued with the remaining drive. Performance measurements were repeated on the remaining drive to obtain the same results for non-mirrored drives. The downed Mirror was then brought back on-line and re-mirrored. The measurements were again checked. In the next phase of the experiment, one of the Mirrored disks was placed on a second controller, and all of the studies were repeated.

The experiment was performed on two databases that resided on separate logical disks on the CDC drives. Each drive was subdivided into three logical disks, so that the position of the target data on the disk could be controlled. The relation used to make the measurements was the "tenktup" benchmark relation defined by David DeWitt in his paper at the University of Wisconsin(1983). This relation contains 10,000 tuples and occupies 1000 disk blocks. Since the IDM had only 450 cache pages, a scan of the relation required 1000 disk reads.

The queries that were measured were queries that were largely disk bound. The queries selected all had measurable amounts of disk reads and writes. Some performed primarily reads, some performed only writes, some did both. The object was to extract the effect of mirroring the disks. Each measurement was repeated several times when possible. Some measurements were made for multi-user conditions when this was relevant.

Each table contains the times necessary to perform the measured function with Mirrored and Non-mirrored disks. The Mirrored disk times are given for both the single controller and the tandem controller cases. While times for the Non-mirrored and single controller cases were fairly consistent on repeated trials (within 1 second in most cases), times for the Mirrored case with separate controllers varied much more. For these studies, the mean time is given, and the observed range follows in parentheses. The percentage difference in time is given by:

$$\text{difference} = \frac{\text{Mirrored} - \text{Non-mirrored}}{\text{Mirrored}} \times 100$$

Negative values for the difference indicate a performance enhancement by Mirrored disks.

IDMCOPY IN

The process of copying a relation into the IDM from tape or host is one that requires considerable disk I/O. In this experiment, the tenktup relation and another, smaller relation ("eds5") were copied from the VMS host. The times were the same for the IDM tape, since this utility is limited by the IDM performance.

Relation	Mirrored(sec)	Non-mirrored(sec)	difference
tenktup (1 controller)	105.	96.	9%
tenktup (2 controllers)	107.5(106-109)	10%	
eds5 (1 controller)	85.	81.	5%

Mirrored disks were slightly slower in recording the relations, most likely due to factors explained above.

RETRIEVE--Relation Scan

A retrieve command that causes the entire relation to be scanned, but does not output large volumes to the host is a good measure of the disk read performance on the IDM. The command used here required 1000 disk reads to scan the relation.

range of t is tenktup
retrieve (c=count(t.unique1));

Controllers	Mirrored (sec)	Non-mirrored (sec)	difference
1	18.7	18.8	0%
2	18.0(17-19)		-4%

No significant difference in performance for the Mirrored and Non-mirrored disks was seen with one controller. With 2 controllers, the Mirrored disks performed somewhat faster most of the time.

CREATE NONCLUSTERED INDEX

To create a nonclustered index, the entire relation must be read and the index structure created.

create nonclustered index on tenktup(unique1);

Controllers	Mirrored (sec)	Non-mirrored (sec)	difference
1	42.7	46.3	-8%
2	43.2(42-45)		-7%

The Mirrored disk configuration was somewhat faster. This operation involves mostly reads of the disk.

CREATE CLUSTERED INDEX

The create clustered index command causes the entire relation to be sorted on disk. The original relation is scanned, a new, sorted relation is written, and then the original is destroyed. The operation requires both CPU and disk reads and writes.

create clustered index on tenktup(unique2);

Controllers	Mirrored (sec)	Non-mirrored (sec)	difference
1	88.1	77.8	12%
2	80.(77-8)		3%

We see the cost of writing to disk in this case. The benefits derived from sequential reading of disk blocks are obscured by interleaved processing and writing of the new relation. Again, the two controller configuration performs considerably better.

RETRIEVE INTO

The IDL *retrieve into* command is actually more demanding than creating a clustered index. This command is considered an update, so that all changes must be written to the batch log or transaction log before the change is performed. It is used as a better indication of disk writes than simple Appends would provide.

range of t is tenktup

Controllers	Mirrored (sec)	Non-mirrored (sec)	difference
1	88.8	79.7	10%
2	85.7(78-91)		7%

REPLACE

Replace, like *retrieve into*, requires all changes to be written to a log before being committed. In this case the log must be read back, after being written to disk and applied to the original relation. The command used in this experiment changes every tuple to its current contents. The cost is the same as if the attribute were altered. The process required 4000 reads and 4000 writes for the 1000-block relation when a non-clustered index existed.

range of t is tenktup
replace t (two=t.two);

Controllers	Mirrored (sec)	Non-mirrored (sec)	difference
1	217.	184.	15%
2	174.(168-180)		-6%

This operation has the largest cost factor for single-controller Mirrored disks of any of those measured. It is also an illustration of a clear improvement in performance for the second controller. The large number of disk operations certainly provides an extreme case for disk performance.

Multi-User RETRIEVES

This is a special case where the presence of the Mirrored disk feature provides a significant enhancement to performance. Three cases were investigated. In each case, two users invoked a command to scan the relation on the disk simultaneously.

In the first case (labeled "Shared"), the two users shared data, that is they were trying to scan the same relation in the same order. Each user profited from the disk operations of the other, so that neither process suffered much in performance.

In the second case (labeled "Same disk"), the users scanned relations on the same disk (Mirrored or Non-mirrored).

In the third case (labeled "Diff disk"), the users scanned relations on different physical disk drives.

For these measurements, the same scan that was illustrated above was run by each user. The times given are the average times for each of the two user processes.

Case/cntrl	Mirrored (sec)	Non-mirrored (sec)	difference
Shared/1	18.6	18.8	1 %
Shared/2	18.6		1 %
Same disk/1	25.6	91.8	- 260 %
Same disk/2	20.0		- 350 %
Diff disk/1	25.6		
Diff disk/2	22.0		

Note that the two users complete their scan in the same time when they are sharing data. The two processes are in many ways symmetrical, each doing half of the disk reads and waiting on the other half.

When the users are scanning data on the same disk, but different locations, they must compete for the same read-write head. Not only must each user wait for the disk to read the competing data, but he must also wait for the seek times between the tracks. The two processes alternate diskblocks, causing excessive seeks in the Non-mirrored case. In the Mirrored case, each Mirror will read the blocks of one user, providing performance identical to having the data located on separate disks.

The "Diff disk" measurement actually had one user scanning the Non-mirrored disk and the other scanning data on the Mirrored disk. The two processes share the CPU and available cache memory, but most processing is overlapped with disk I/O for the other job.

Other Possible Factors for Investigation

Some experiments have not been performed in this study:

Multiple user queries may provide a mix of the performance seen, but they have not been explicitly investigated.

Mirrored system disks may give different results. It is possible that the cost of writing system relations may add to all queries. This too will be the subject of a later investigation.

IDM Performance Test
Relative Performance of 10 and 6 MHz Database Processors

Ed Simon

Overview

In order to express the relative performance of the 10 MHz database processor in terms of the current 6 MHz processor, six queries were performed and the effective transaction rate for each query calculated.

For each query, two types of comparisons were made. First, the effect of the DAC on the query performance was calculated for both the 6 MHz and 10 MHz cases. Second the percent improvement of the 10 MHz over the 6 MHz DBP was calculated.

As might be expected, the improvement of the 10 MHz DBP over the 6 MHz DBP was greater without the DAC. The improvement with the DAC was greatest for those queries that made least use of the DAC.

For these 6 queries, the average performance improvement of the 10 MHz DBP over the 6 MHz DBP was 59% without the DAC, 52% with the DAC.

For the first 3 queries, the 6 MHz/DAC was faster than the 10 MHz/NODAC. For the last 3 queries, the 10 MHz/NODAC was faster than the 6 MHz/DAC.

Benchmark Conditions

Each experiment was performed on an IDM with 2 Megabytes of memory and:

6 MHz/DAC – 6 MHz DBP with the database accelerator

6 MHz/NODAC – 6 MHz DBP without database accelerator

10 MHz/DAC – 10 MHz DBP with the database accelerator

10 MHz/DAC – 10 MHz DBP without database accelerator

All times were taken from set 5 times on queries run on the onektup relation defined by DeWitt in his 1983 benchmark. The relation (100 disk pages) was cached before the standalone series of queries. Per-process performance monitoring was used to insure that no disk I/O or host I/O contributed to the elapsed times. The queries were repeated several times in four configurations, and found to vary little with repetition. The set 5 time for each query was recorded in ticks (60ths of a second) and the transaction rate calculated by:

$$\text{rate} = \frac{60}{\text{set 5 (ticks)}}$$

The percent 10 MHz improvement was then found by:

$$\text{percent} = \frac{\text{rate(10MHz)} - \text{rate(6MHz)}}{\text{rate(6MHz)}} \times 100$$

The percent DAC improvement was then found by:

$$\text{percent} = \frac{\text{rate(DAC)} - \text{rate(NODAC)}}{\text{rate(DAC)}} \times 100$$

Summary Results

Query 1: Scan relation searching integer field.

retrieve (c=count (o.two where o.unique2=488));

Effective Transaction throughput for this query:

	NODAC	DAC
6 MHz	100%	333%
10 MHz	150%	500%

DAC Improvement = 233% (6 MHz), 233% (10 MHz)
10 MHz Improvement = 50% (NODAC), 50% (DAC)

Query 2: Scan relation searching on string field

retrieve (c=count (o.two where o.stringu2="G"));*

Effective Transaction throughput for this query:

	NODAC	DAC
6 MHz	100%	385%
10 MHz	162%	524%

DAC Improvement = 285% (6 MHz), 223% (10 MHz)
10 MHz Improvement = 62% (NODAC), 36% (DAC)

Query 3: Aggregate function on integers

retrieve (c=avg (int4(o.thousand) by o.ten));

Effective Transaction throughput for this query:

	NODAC	DAC
6 MHz	100%	181%
10 MHz	155%	268%

DAC Improvement = 81% (6 MHz), 73% (10 MHz)
10 MHz Improvement = 55% (NODAC), 48% (DAC)

Query 4: Aggregate function with bcd conversion

retrieve (c=avg(bcdflt(0,o.thousand) by o.ten));

Effective Transaction throughput for this query:

	NODAC	DAC
6 MHz	100%	133%
10 MHz	158%	208%

DAC Improvement = 33% (6 MHz), 32% (10 MHz)
10 MHz Improvement = 58% (NODAC), 56% (DAC)

Query 5: Scan relation searching on string with lead wild card.

*retrieve (c=count(o.two where o.stringu2="*G*"));*

Effective Transaction throughput for this query:

	NODAC	DAC
6 MHz	100%	119%
10 MHz	170%	200%

DAC Improvement = 19% (6 MHz), 18% (10 MHz)
10 MHz Improvement = 70% (NODAC), 69% (DAC)

Query 6: Scan relation using OR clause to force sort of tuple ids.

retrieve (c=count(o.two where o.two=1 or o.two=0));

Effective Transaction throughput for this query:

	NODAC	DAC
6 MHz	100%	139%
10 MHz	159%	217%

DAC Improvement = 39% (6 MHz), 37% (10 MHz)
10 MHz Improvement = 61% (NODAC), 56% (DAC)

Detail Results

QUERY TIMES (60ths of a second)

Query	6MHz/NODAC	10MHz/NODAC	6MHz/DAC	10MHz/DAC
1	30	20	9	6
2	42	26	11	8
3	267	174	151	102
4	492	315	375	243
5	101	60	86	51
6	65	41	47	30

QUERY TRANSACTION RATES (Queries/second)

Query	6MHz/NODAC	10MHz/NODAC	6MHz/DAC	10MHz/DAC
1	2.0	3.0	6.7	10.0
2	1.43	2.31	5.5	7.5
3	0.22	0.34	0.40	0.59
4	0.12	0.19	0.16	0.25
5	0.59	1.00	0.70	1.18
6	0.92	1.46	1.28	2.00

Conclusion

It can be seen that the overall range in processor performance indicated by these studies is between a factor of 2 and a factor of 5. In some cases, the Database Accelerator provides considerably more throughput enhancement than the faster DBP; in others the 10 MHz DBP actually provides a somewhat larger enhancement than the DAC. Taken together, the two high performance processors can endow the IDM 500 with considerably better performance for query processing.