Virtuoso and Database Scalability

By Orri Erling

Table of Contents

Abstract

This article shows results of running a TPC-C-based benchmark on a number of different platforms with different CPUs, amounts of memory, and numbers of disks. The motivation for the tests was finding a suitable building block for a Virtuoso-based web application server farm.

We concentrate on hardware and database scales that are most likely relevant for developers of small to mid-size online applications. The database size starts at 4G, and the test has 80 concurrent clients. We compare systems with between 512M and 4G of memory, and 1 to 4 disks.

The subject of benchmarking and optimizing the operation of a database server on a particular OS for a particular workload is extremely complex. In this article, we show some sample results and outline dependencies between factors affecting performance but do not claim to present an exhaustive study of the subject.

The numbers obtained are not comparable with the official TPC-C metric. The type of system and the test rules are quite different.

Metrics

The benchmarks measure the following:

- Time to load a 40 warehouse TPC C database. This is about 4G worth of short data rows, inserted in ascending order from an ODBC client application. This measures the speed of a single CPU, as the operation is mostly CPU bound with serial disk writes taking place in the background. Before this is done, a 12GB empty database is preallocated, striped on as many disks as are available.
- The transactions per minute metric, running 80 clients, 2 per warehouse, each client producing the TPC-C transaction mix, starting the next transaction as soon as the previous completes. This is affected by a combination of random disk access latency, amount of memory and CPU speed.
- Serial read - This measures the disk throughput for reading through a large fragmented database table, a typical operation for any business intelligence query.

Results

Transaction Throughput

Throughput	disk type and quantity	RAM	processor kind, speed, and quantity	CPU%	System
19001	3 x SCSI	44 GB	2 x SPARC, 1.6GHz	120%	Solaris 10 on SunFire 440
11580	4 x SATA	1 GB	1 x AMD64, 2 GHz	50%	FC4 on AMD64 Generic PC
8138	1 x SCSI	2 GB	2 x AMD64, 1.8GHz	42%	Solaris 10 on SunFire V20z
6081	4 x SCSI RAID	2 GB	4 x Pentium 3 700 MHz	N/A	Windows 2000 on Dell
2989	2 x SATA	1 GB	1 x AMD64 2GHz	10%	FC4 on Generic PC
2701	4 x SCSI	0.5 GB	2 x Pentium 3 600MHz	102%	Solaris 7 on Generic PC
83937	4 x SATA 7200rpm	8 GB	2 x dual core Xeon 5130, 2GHz, 64 bit mode	274%	Supermicro 7045 A/T
49450	4 x SATA 7200rpm	8 GB	2 x dual core Xeon 5130, 2GHz, 64 bit mode	187%	Supermicro 7045 A/T
2607	1 x 80G built-in	2 GB	1 x Intel Core Duo 1.66GHz	27%	Mac OS X on Mac mini

The throughput is the count of successful new order transactions made during the test run, scaled to a transactions per minute count. For every 10 new orders, there are 10 payments, 1 delivery, 1 order status, and 1 stock level transaction. These are not separately reported.

The CPU% is percentage of one CPU; thus on multiprocessor systems, the percentage can be over 100.

Initializing 40 warehouses

Time	disk type and quantity	processor kind, speed, and quantity	System
12m 6.874s	4 x SATA	1 x AMD64 2GHz	FC4 on AMD64 Generic PC
17m 4.007s	3 x SCSI	2 x SPARC 1.6GHz	Solaris 10 on SunFire 440
17m 50.294s	1 x SCSI	2 x AMD64, 1.8GHz	Solaris 10 on SunFire V20z
44m	4 x SCSI	2 x Pentium 3 600MHz	Solaris 7 on Generic PC
4m 34s	4 x SATA 7200rpm	2 x dual core Xeon 5130, 2GHz, 64 bit mode	Supermicro 7045 A/T
14m 14s	1 x 80G built-in	1 x Intel Core Duo 1.66GHz	Mac OS X on Mac mini

Serial Read

MB per second	disk type and quantity	System
8.25	3 x SCSI	Solaris 10 on SunFire 440
7.77	4 x SATA	FC4 on AMD64 Generic PC
2.69	1 x SCSI	Solaris 10 on SunFire V20z

Test Conditions

Each system was tested with the database and 80 clients running on the same machine. First a 12G preallocated test database was made, then filled with 40 warehouses. After this, 80 test drivers were started, two per warehouse, and left to run for 60 minutes of real time. After 60 minutes the database was shut down, disconnecting the clients. The database was restarted and the number of new orders was counted. This number minus the original number of orders divided by 60 is the metric. After this the sequential read test was run.

Below are the relevant lines of the virtuoso.ini file used for the test:

```
[Database]
TransactionFile = tpcc.trx
Striping        = 1

[Parameters]
ServerThreads        = 1000
;                               These are allocated as needed,
the essential
;                               is to have 30% more than the
number of
;                               concurrent clients.
CheckpointInterval   = 20
```

```
;                                 To have a steady state test, one
that does
;                                 not accumulate an infinite
transaction log,
;                                 we checkpoint every 20 minutes,
three times
;                                 per the one hour test run.
NumberOfBuffers      = 43000
;                                 43000 if 512MB RAM; 120000 if 2G
- Systems
;                                 were tried with different
settings and the
;                                 best result was taken.
MaxDirtyBuffers      = 30000
;                                 About 3/4 of number of buffers
MaxCheckpointRemap   = 500000
;                                 Make this larger than the working
set, so
;                                 that checkpoints will just be
buffer flushes
;                                 without need to move data inside
db files.
;                                 500000 pages is about 4G
FDsPerFile = 16
;                                 Especially if the database is a
single file
;                                 or few files, especially if on a
striped
;                                 RAID, this should be large.  This
controls
;                                 how many threads may have a
read/write
;                                 system call going on a single
file.  The OS
;                                 will get to sort them and/or do
them in
;                                 parallel if the fs is striped.
It is best
;                                 to do striping at the db level
and to have
;                                 disks that are known to be
independent rather
;                                 than relying on a RAID controller
for this.

[Striping]
Segment1 = 12G, tpcc1.seg q1
;                                 Add files here, one file per
independent disk,
;                                 have q2...qn after each to
allocate dedicated
;                                 I/O thread per device
```

Analysis

Working Set

To understand the dynamic between main memory and I/O, we have to look at the working set of the transaction mix. The stock and customer tables amount to about 58 MB per warehouse and are subject to constant updating. The frequency of access to all rows is not equal but we may idealize the situation by assuming that at the page level all pages of these tables get even frequency of access even if the rows do not. Additionally, there are 10 points per warehouse in the `*orders*` and `*order_line*` tables where sequential inserts take place. Both the delivery and stock level transactions concern recently inserted orders. The delivery transaction updates orders 2.8 MB worth of inserts after they were first inserted. If we count new orders before they get updated by the delivery transaction and are subsequently not touched as part of working set, we get an extra 28 MB of working set per warehouse. With low memory, order lines are not likely to stay in cache for the interval between initial insertion and update on delivery, so we can roughly estimate the per warehouse working set to be 66MB, counting 8 MB for the 40 insert or update points corresponding to insert in `*orders*` and `*order_line*` and the recent order lines scanned by the stock level transaction.

For 40 warehouses, this gives a working set of 2640MB. Since the activity on orders and order lines is localized and constant, the cache misses are divided between the stock and customer tables.

Effect of Server Cache Size

On a system with 2G RAM, we have about half the working set in memory if we configure half the space for database buffers and the other half will go mostly towards OS file caching. If we configure most of the space for database buffers, the OS will swap database buffers out in order to keep more file cache. This is very bad for performance.

On a system with 1G, we may have up to a quarter in memory, again if we configure half the memory for database buffers.

Means of avoiding double buffering are system dependent and are not addressed in this article but may be revisited in a follow up.

We have run the test on a SunFire V20z Solaris 10 machine with 2G of RAM with different amounts of database buffers. The measurements were --

Buffers	Throughput
43000	3872
86000	6955
100000	7819
120000	8137
150000	7253

There are two factors explaining the large effect of the amount of server disk cache:

- There is a difference between having the data in warm OS file cache and having it in the user space of the process. For a `SELECT COUNT (*)` doing a full table scan, having the pages in the server cache as opposed to copying them from warm OS file cache to the server cache makes approximately a threefold difference, measured on Linux.
- Having more buffers in the server cache significantly reduces the number of disk writes per transaction. With 43000 buffers, we have typically 3 disk writes for every new order. With 140000 buffers, we have about 1.4 writes per new order. Additional tuning could drop this still further.

When the amount of main database cache goes over 2/3 of main memory, performance drops sharply, mostly due to the OS swapping out the database process in favor of OS disk cache. Wiring down the database process may help but results will depend on the OS.

CPU Load

The CPU load due to the clients is minimal, about 1/20 of the CPU usage of the server. The clients do nothing except execute stored procedure calls one after the other and do virtually no other I/O.

We see that the database load time essentially reflects the clock speed. This is a single CPU, CPU bound operation where all I/O is sequential background writes.

The transaction rate on the other hand reflects the number of independently addressable disks and the amount of memory.

The split between user and system CPU time was around 15% system CPU, as percentage of total CPU. The percentage rarely exceeded 25%. This was the case in all operating systems.

To find out the difference between CPU bound and I/O bound situations, we decreased the working set to 10 warehouses, to about 650MB and we ran the same test with again two clients per warehouse. After the test reached a steady state after a ramp-up of 7 minutes on our dual AMD64 SunFire, we measured CPU at around 70%,, varying between 80% and 60%, meaning around one and a half of 2 CPUs on the Virtuoso server process. We continued to have 350% disk reads, meaning an average of 3.5 read system calls pending at each time, down from over 20 pending at a time when we ran with the 40 warehouse configuration. We note however that writes to disk do occur, both for transaction log and dirty buffers, even though the frequently updated working set tends not to get written to disk and at any time about 2/3 of the buffers are dirty. Thus the test is not free of I/O; however, it is mostly CPU bound.

The mostly memory based database throughput was 42748 transactions per minute, with 64%

CPU, that is 128% of one CPU. This is about five times the throughput of the heavily I/O bound scenario with the 40 warehouse working set.

Disk Parallelism

If the working set is significantly larger than the server's disk cache performance grows near linearly with the number of disks, at least between 1 and 4 disks.

We see nearly the same throughput for an old 600MHz dual Pentium 3 with 512 MB and a new 2GHz AMD64 with 1GB and 2 SATA disks. In both cases, the benchmark is heavily I/O bound, thus the double amount of parallel disk seeks compensates for having only half the memory.

The fragmented sequential read metric also reflects the number of disks. There is a per-disk ordered read ahead, done with one thread per device, so that the throughput is roughly linear to the number of disks, at least at the beginning. This is independent of the amount of memory, as the read is done in into a cold cache and no row is visited twice. There is substantial fragmentation, thus this should not be compared with raw disk data transfer rates.

Implications for Web Applications

The TPC C workload is I/O intensive. Each transaction requires fairly little CPU in comparison with composing a typical dynamic web page, for example. When profiling the Virtuoso web applications suite, it is not uncommon to have about 15% of the work database related and the rest having to do with composing the page.

TPC C is not representative of an e-commerce application, since most of the clicks processed by one have to do with browsing the catalogue or filling the shopping cart, as opposed to processing an order.

It is safe to estimate that a typical web application, for the same amount of I/O as is generated by TPC-C, will use 5-10x more CPU. Thus, while there was little or no benefit from multiple CPUs in the cases covered, a web application is likely to benefit from 2 or 4 CPUs for the same amount of I/O.

Other DBMS

The same observations will apply to basically any DBMS. We have ported the Virtuoso TPC-C test driver and stored procedures to Oracle and Microsoft SQL Server. The Oracle stored procedures can be found in `binsrc/tests/oracle_tpcc`. The DB2 and Microsoft SQL Server procedures can be found in any TPC full disclosure report. For the interested, making the tests run on any database with stored procedures and ODBC should be a simple task. In many cases, license terms prohibit us from publishing results of tests run on databases other than our own.

Conclusions

From the measurements we infer that the lowest cost per throughput would be found with an AMD64 PC with 4GB memory and as many SATA disks as possible. While high end disks provide better seek times and transfer rates, these are offset by having a larger number of independently seekable units, whether this were for sequential or random access. A commodity PC with this spec costs around $2000-2500, where as the Sun servers tested are more expensive. We do not have exact prices for all the systems tested.

An in depth study of scalability would have to involve measurements with system specific OS tuning for eliminating double buffering and preventing swapping of the server process. Also the performance per added disk should be graphed, up to the point of diminishing returns. Indeed, each answer gives rise to more questions.

We trust however that the data presented herein gives an initial feel for the factors affecting Virtuoso performance on common hardware configurations.