

Jim Gray with help from Bryan Bouma @ sun and Alainna Wonders @ JHU
17 January 2007

Performance of Sun X4500 at JHU (Thumper.pha.jhu.edu) under Windows, NTFS and SQLserver 2005

Executive summary: This is a balanced system with 16GB of fast memory, 4 fast processors, and enough disks and disk bandwidth to keep them busy when doing OLTP or data mining.

Sun Microsystems loaned an X4500 to the Johns Hopkins University Physics Department in Baltimore, Maryland to do Windows-SQLserver performance experiments and to be a public resource for services like SkyServer.org, LifeUnderYourFeet.net, and CasJobs.sdss.org. The system's applications will soon be on the Internet, but for now we are just testing it. I *termserved* to Thumper.pha.jhu.edu from San Francisco.

The X4500 is a quad-core 2.6GHz Opteron, with 16GB of RAM, 6 Marvell SATA controllers, 48 500GB Hitachi 7krpm SATAII disks, and dual Gbps Ethernet in a 4U package consuming about 0.8 KW when all the disks are working hard.

As a first step we installed Windows 2003 Standard SP1 in 64-bit mode, Visual Studio 2005, and SQLserver 2005 Standard (64bit) SP1.

An earlier study showed that this system using a JBOD (just a bunch of disk == Raid0) configuration with 10 GB test files on each disk delivered 8,689 random 8KB reads/s and 13,080 random 8KB writes/s with a 160-deep queue and 2,670 MB/s sequential read and 1,881 MB/s sequential writes. The small files (10GB) were mandated by the short test period available and so the seeks were across 2.5% of the disk. As seen below, ~100 IO/s/disk rather than the ~200 IO/s/disk implicit in the random IO numbers should be expected if seeks must cross 1/3 of the disk surface on each request – i.e. if most of the disk space is used for randomly accessed files.

This time the 48 disks were mirrored across the 6 controllers to create 24 dynamic volumes (we chose mirroring rather than JBOD because this will be a showcase service and we avoided RAID5 because it has given us so much trouble in the past). This decision gives up about 1/2 the disk capacity (12TB vs 24TB) and about 1/2 the write IO bandwidth (mirrored writes cost 2 accesses). But it buys peace of mind. Advanced write combining (write cache) was enabled for each volume.

Using memspeed.exe, single processor memory reads at 3.0 GB/s, writes at 1.7 GB/s, and copies at 1.3 GB/s. Using SSE, 4 processors, read at 6.4 GB/s, write at 8.9 GB/s, and copy at 4.3 GB/s

The first IO tests measured the sequential and random performance of the system. We created a 400GB file on each volume and then used [SqlIo.exe](#) to generate sequential or random read and write traffic against that array, first one disk at a time and then a ramp of 2, 3, ..23 volumes (46 disks) in parallel. The resulting throughput is cartooned in Figure 2 (a screenshot from Perfmon.)

The graph suggests that each disk delivers about 100 random iops while reading and nearly 150 iops when writing (likely due to the disk cache).

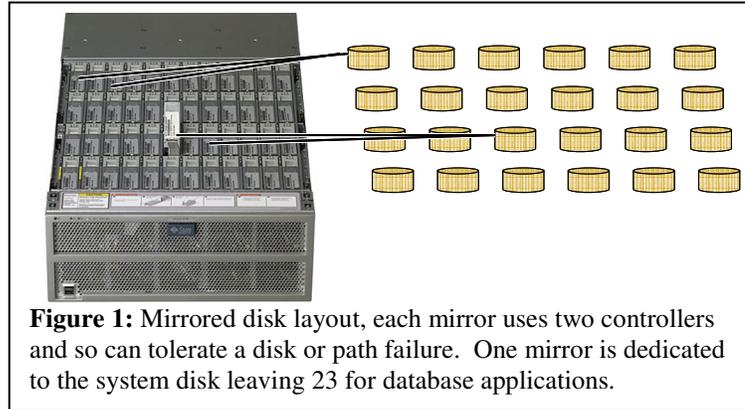


Figure 1: Mirrored disk layout, each mirror uses two controllers and so can tolerate a disk or path failure. One mirror is dedicated to the system disk leaving 23 for database applications.

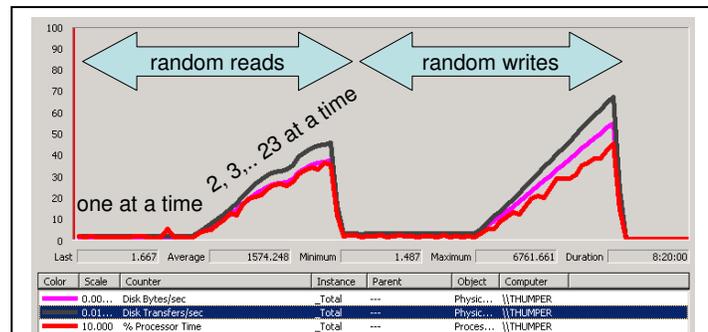


Figure 2: A perfmon graph of 8-deep random IO 8KB. Red is cpu (x10), pink is MB/s and black is transfers/second/100 (peak at right is 6.6K iops, 50MBps, and 4% of 4cpus = 15% of one cpu.) These are physical IOs so the mirrored writes actually run at 3.3k logical iops and 25MBps at the application level.

There are no bottlenecks in the memory or controller (at 6.6k iops there are 3.6 cpu's available to the application.) Figure 3 graphs the measured read and write and the attached spreadsheet has the actual numbers. The second part of Figure 4 shows the sequential IO behavior of a single mirrored volume for various queue depths and block sizes. Averaged across the surface, the disks deliver ~55MB/s, the outer edge is 65 MB/s and the inner tracks are 44 MBps.

In between these experiments we ran a torture test doing over 200 hours of small random IOs. One of the 48 drives (H:) started misbehaving (writing very slowly) during this process and is being retired.

Figure 3: The graph at right shows the random IO/s of Figure 2. Each experiment was run for 5 minutes using SqlIO.exe with 8KB requests keeping the queue to each volume 8-deep. The “bump” in the read throughput at about 12 drives seems to be real (repeated many times). These numbers are lower than reported in a previous report because the seeks are across the entire disk (each has a 400GB file) whereas previous experiments used files 40x smaller and so had much quicker seeks. Your random IO performance is guaranteed not to exceed these numbers 😊 unless you have smaller files. At peak the array is delivering ~3.5k writes per second or 4.6k reads per second. Only one of the 4 cpus is busy at the peak – leaving a lot of processing power for applications.

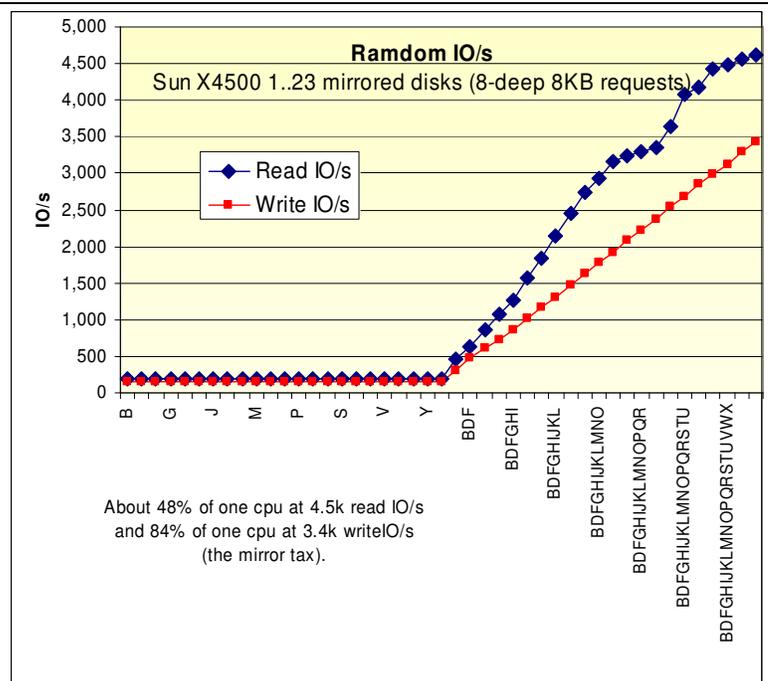
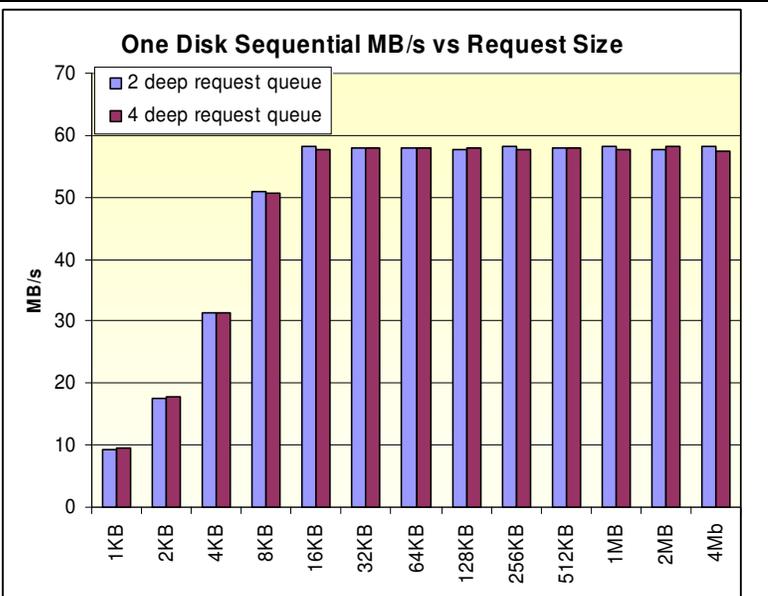


Figure 4. The one disk sequential bandwidth peaks at about 57MB/s and is approximately the same for reads and writes. The disk achieves nearly full bandwidth with fairly small requests (16KB) and fairly shallow queues (2-deep).



The following two figures show the array's sequential IO performance using 2-deep 256KB requests. The figure captions explain the diagrams and results.

Figure 5. A perfmon screen shot of the bandwidth x10MB/s (white line), the physical IOs x100 (black line), and the cpu consumption (% of 4 cpus red line) for the experiments graphed below. First each individual disk is tested for a minute and then the growing array of discs is tested for a minute each. It shows that the cpus are nearly idle (4x20% at most == 80% of one of the 4 cpus). At peak the system is reading at 1.09 GB/s and writing at 2.14 GB/s (mirrored writes). The application only sees 1/2 that write bandwidth.

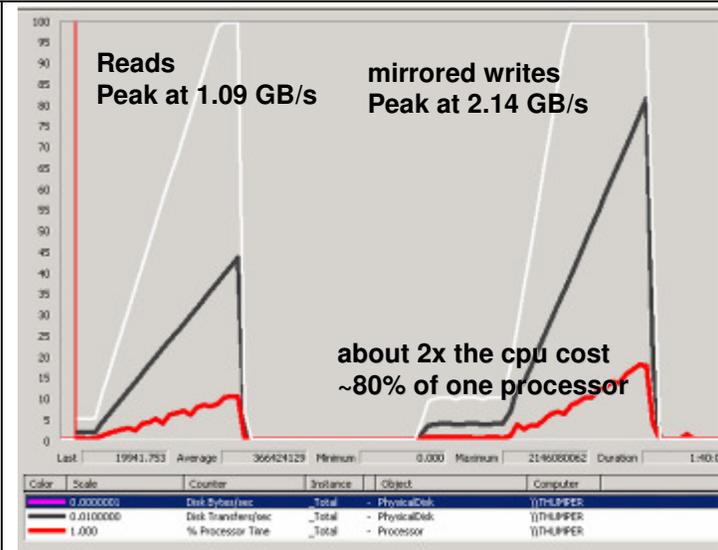
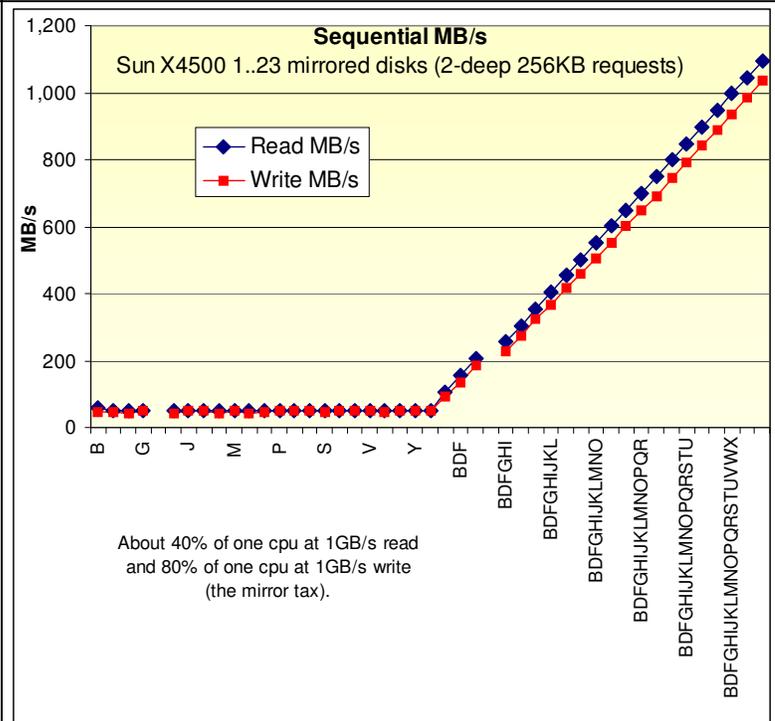


Figure 6. The same data as Figure 5 showing the sequential disk read and write bandwidth using SqlIO. The request queue was kept 2-deep and requests were 256KB. The read bandwidth could be about twice the write bandwidth since the file system can read from both disks in parallel while each write must write both disks in the mirrored pair. The Windows 2003 volume manager does not have that optimization and so read and write bandwidth are comparable. Again, when writing ~2GB/s the system has 3.2 cpus available to applications. The tests do not include the "H" volume which failed (slowed down) during the experiments.



SQL testing – transaction processing

We created the simple *TheBank* database described in [“Thousands of DebitCredit Transactions-Per-Second: Easy and Inexpensive,”](#) MSR-TR-2005-39 consisting of 10M bank accounts (equivalent to a very large bank). The database uses one file group striped across 22 volumes (44 disks). The DebitCredit benchmark was run against it using 1, 2,... and more threads. Figure 6 cartoons the throughput. With about 8 request streams the system is processing ~9k transactions per second (peak of 11k tps) and using about 30% of the 4 cpus – it is bottlenecked on the log force (every SQLserver transaction forces the log.) The big pink spike after about 10 minutes is checkpoint IO. Two generations of database people spent their careers optimizing this benchmark. One early optimization addressed the log force bottleneck by introducing *group commit*.

Very few people need more than 9Ktps – that would be a VERY busy time for the whole Visa credit card system which involves many computers. That may explain why SQLserver lacks group commit. But, group commit is easy to implement, and indeed Erik Christensen invented a nice adaptive algorithm and prototyped it in a SQLserver.exe (Eric’s change is not going to appear in the product anytime soon, it is just for experiments.)

Figure 7 shows the performance of Eric’s group-commit implementation as we ramp up to 80 threads driving the server. At about 18 threads, the system is delivering about 20k tps and using about 80% of the cpu. As more threads are added, the number of log writes per second actually declines to about 20/s (this is Eric’s algorithm kicking in.) Indeed the whole scale for the log writes was reduced by 10x so that it shows well in this graph compared to Figure 6 – rather than 13k log writes per second it is doing more like 20-40 log writes per second and of course doing 3x more transactions per second in the process.

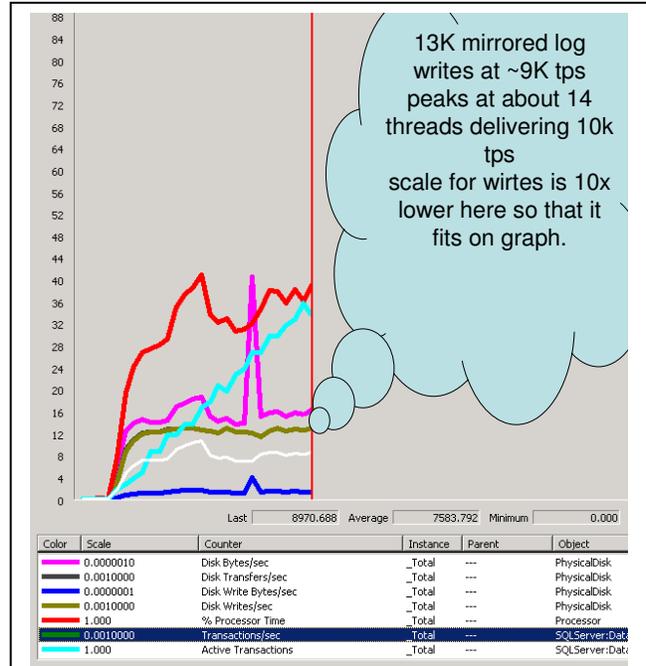


Figure 6: The out-of-the-box performance of SQLServer on the DebitCredit transaction.

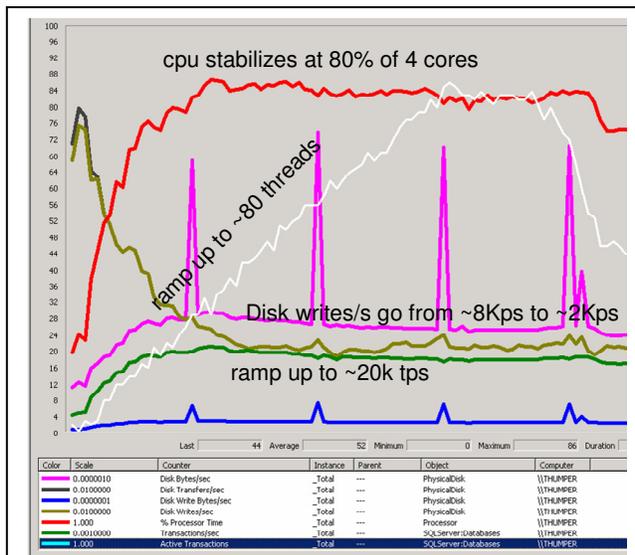


Figure 7: The performance of SQLServer on the DebitCredit Transaction using Eric Christian’s group commit algorithm.

SQL testing – transaction processing

Next we created four tables, ten, hundred, thousand, and page, that had respectively 10-byte, 100-byte, 1KB, and 8KB records. We populated these tables with 2B, 1B, 100M, and 10M rows giving us a 36GB, 105GB, 96GB and 77GB tables respectively (see Table 2).

	GB	rows	bytes/record
ten	36	2B	19
hundred	105	1B	112
Thousand	96	100M	1,029
page	77	10M	8,232

We then ran the following two queries against these tables using read-uncommitted transaction isolation. We use `count_big()` to avoid overflowing the counter (`count()` overflows at 4B).

trivial predicate

```
select count_big(*)
from table
where a = 0
```

complex predicate:

```
select stdev(
  sqrt(
    sin(a*pi())*sin(a*pi())
    +cos(a*pi())*cos(a*pi())
  )-1.0)
from table
```

Table 3: Times and throughput for trivial and complex predicates.

Trivial predicate						
	scan time	cpu time	cpu%	cpu us/rec	Mrec /sec	MB/s
Ten	96	235	245%	0.12	21	415
Hundred	157	209	170%	0.21	6	710
Thousand	141	127	150%	0.13	0.7	750
Page	111	91	120%	9.15	0.1	750
Complex predicate						
	scan time	cpu time	cpu%	cpu us/rec	Mrec /sec	MB/s
Ten	1,746	6,956	398%	3.48	1.1	11
Hundred	917	3,593	392%	3.59	1.1	117
Thousand	209	440	210%	0.44	4.8	514
Page	112	108	96%	0.11	0.9	742

The first statement tries to saturate the disks' sequential bandwidth the second statement tries to saturate the cpus (all 4 of them). Figure 8 shows the results graphically. It is possible to keep all four cpus busy. Table 3 translates Figure 8 into numbers. The cpus can read 21M records per second. The database can drive the disks at 75% of their rated speed – 750MB/s.

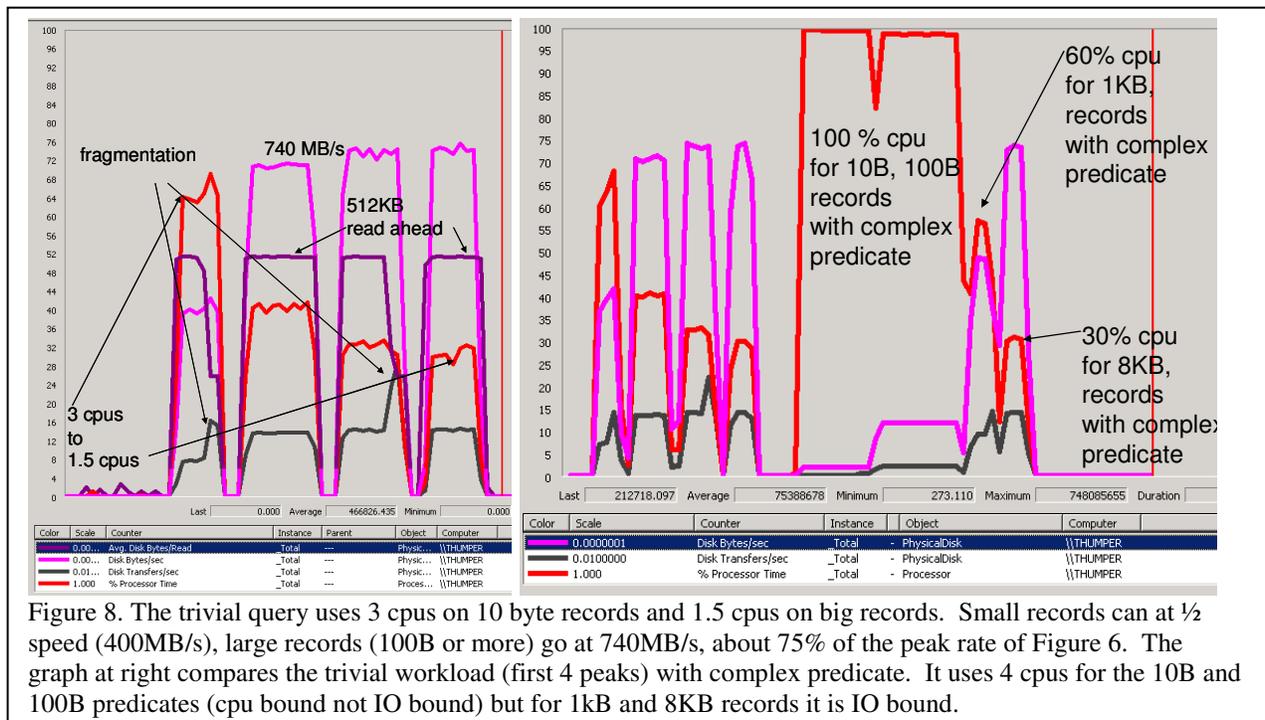


Figure 8. The trivial query uses 3 cpus on 10 byte records and 1.5 cpus on big records. Small records can at 1/2 speed (400MB/s), large records (100B or more) go at 740MB/s, about 75% of the peak rate of Figure 6. The graph at right compares the trivial workload (first 4 peaks) with complex predicate. It uses 4 cpus for the 10B and 100B predicates (cpu bound not IO bound) but for 1kB and 8KB records it is IO bound.

Summary

This system has great memory bandwidth, excellent IO bandwidth, lots of IOs/second, and seems both compact and relatively power-efficient.

As a rough rule of thumb, when configured as a JBOD, the raw hardware delivers

- 2.6 GB/s sequential read,
- 2.2 GB/s sequential write, and
- 5K random reads or writes per second on big files and 9K random IO/s on small files.

When configured with Windows native RAID1 (mirroring) in 23 volumes it delivers about ½ that performance:

- 2.6 GB/s sequential read,
- 2.2 GB/s sequential write,
- 4.6K random 8KB reads per second, and
- 3.5K random 8KB writes per second.

We want to configure fewer RAID10 volumes (6 rather than 24) but NTFS does not support that (it supports only mirrors (RAID1) or JBOD (RAID0)). . We will install the Veritas (Semantic) product that adds RAID10 support to NTFS. The system will have a mirrored system drive and five RAID10 2x4 volumes, and one RAID10 2x3 volume using the Veritas software. These experiments will be repeated when we get that configuration.

SQLserver 2005 delivers about 9K tps on the DebitCredit benchmark and is log-limited. When group-commit is added to SQLserver 2005, it cpu saturates at about 20k tps.

Sequential scans run at 750MB/s if the records are 100 bytes or more and the predicates are simple. If the records are small (10 bytes) then the scans are cpu limited and the system runs at 21 million records per second (the 5M records/second/cpu we see in other benchmarks). SQL is able to use all the cpus if the records are short or if the query is complex. In particular a very complex predicate can be cpu limited for records of 100B or even 1KB.

This is the fastest Intel/AMD system we have ever benchmarked. The 6+ GB/s memory system (4.5GB/s copy) is very promising. Not reported here, but very promising is that we repeated most of the SkyServer Query log analysis on this system – performance was 3x to 100x what we experienced on previous systems – largely due to the 64-bit SQL and to the 16GB of RAM. We hope to report the SkyServer query results soon.

Our next steps are to reconfigure the disk array to be a few volumes and then install the SkyServer DR5, CasJobs, SkyServer Weblogs, LifeUnderYour feet, and perhaps some other portals on this server. We expect it to be on the public internet so that scientists can use it and so that people can experiment with it.

Acknowledgements

Greg Papadopoulos and Jason Woods arranged for the loan of this system to JHU for testing and as the basis for some eScience Internet services. We are VERY grateful to them and to Sun for that. Microsoft donated software and some support. John Fowler, Ben Lenail, and Pete May of Sun facilitated this process. Jan Vandenberg, Alex Szalay, and Ani Thakar of JHU are intimately involved in this project and provided advice all along the way. Catharine van Ingen helped improve the presentation.