Performance Characterization of Disc Caching: A Case Study

Becky McBride & Sam Boles

Hewlett-Packard

Synopsis

Disc Caching is one of the most significant performance enhancements to be introduced in the HP3000 line. The basic theory is clearly sound, but even in the conceptualizing stage the development team knew that empirical validation was essential before announcement of the product.

As soon as the code stabilized after integration, the Cupertino Systems Performance team moved in to subject it to an intensive 4-month performance characterization exercise.

The performance team analyzed existing benchmark test sets and selected a range of database and sequential file tests. This approach gave a real-world flavor to the test sets, and leveraged on the substantial engineering resources already invested in performance benchmarking.

The team then created an environment in which they emulated terminal load by driving the system under test with terminal messages generated by another computer direct-connected to the system under test. This enabled not just a measurement of resource utilization on the system under test, but also a measurement of response times and transaction throughput time at the individual "terminal" level. The automation of the terminal load also enabled precise repeatability and exactly controlled think time and typing speed -- all critical to the measurement of sensitivity to memory pressure, disc configuration changes, number of concurrent sessions and other variables.

The joint effort of the development team and the performance team resulted in a definitive performance profile of the product prior to manufacturing release.

Background: The Other 2 1/2 Rule and the Bottom Line

The Other 2 1/2 Rule

Back in the Dinosaur Days of computers - 20 years or so ago - somebody at the Pentagon came up with the "2 1/2 Rule." They had had a lot of experience with the giant, monolithic systems contracts for the Defense Department. The Rule went something like this: Big computer systems take 2 times as long to build as planned, cost 2 times as much, and have 1/2 the functions and performance.

Any of us with combat experience building big systems (or even little ones) can empathize with the 2 1/2 Rule. We might debate the numbers (they may be too optimistic), but the message is painfully true.

There's another 2 1/2 Rule in the computer industry. It's a little more positive. It says: Every 2 years deliver twice the performance or cut the price in half -- or do both.

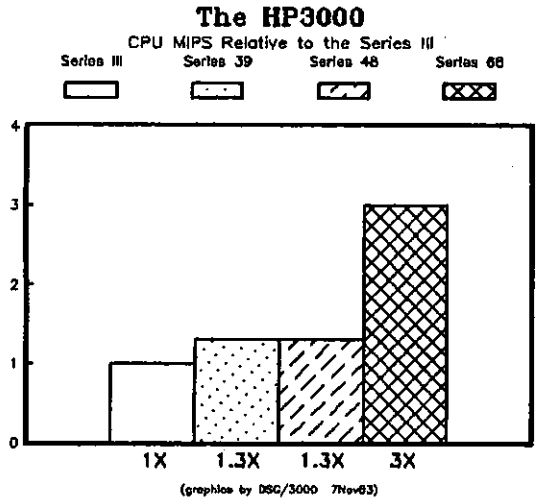
The 2 1/2 Rule and the HP3000

We've all seen the good 2 1/2 Rule in action in the HP3000 family. It's a growing family in performance and functionality and it's competitive price-wise in an industry whose amazing productivity contribution is surpassed only by it's dramatic price competitiveness.

A Few Surprises

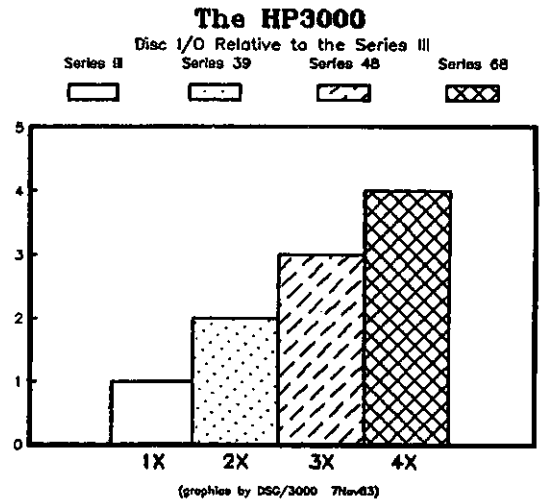
We've seen some impressive advances in processing power, main memory capacity, I/O rates and other system components. And we've seen performance gains. But we've had a few surprises.

Let's take a look at what's happened.



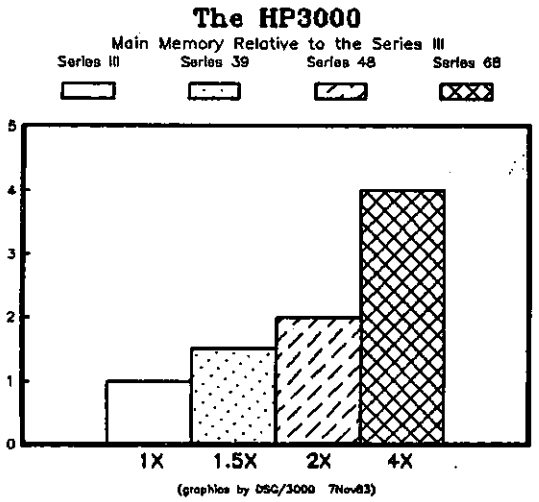
. . . With MIPS Like These . . . .

In the area of CPU (Central Processing Unit) power, we've seen the MIPS (Million Instructions Per Second) grow by more than a 3X factor as we moved from the Series III (about 0.3 MIPS) to the Series 64 and 68 (1.0 MIPS).



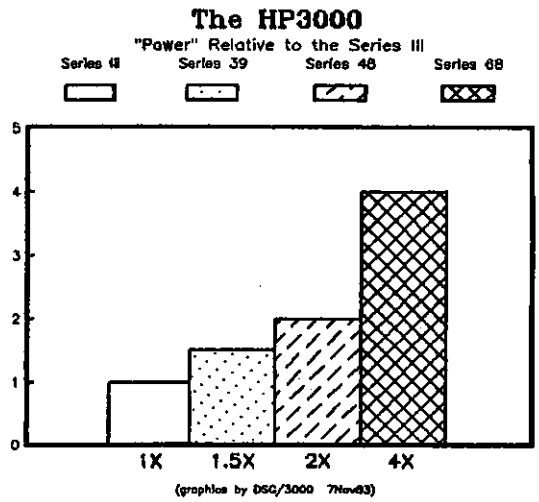
. . . and Disc I/O Rates Like These . . . .

From the 30 or 40 disc I/O's per second on the Series III we've grown to the 120-160 (and higher) range on the Series 64 and 68.



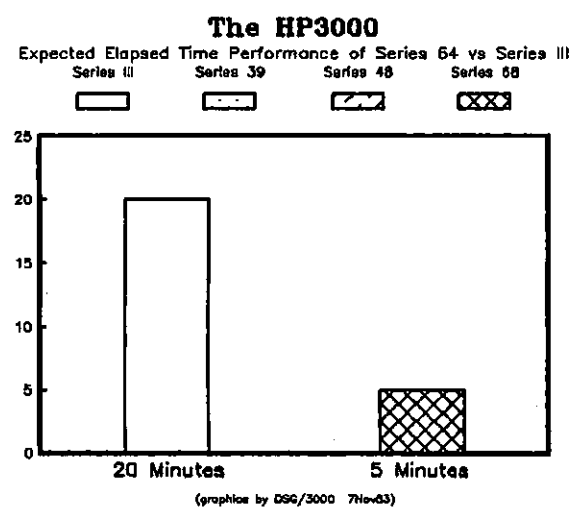
. . . and Main Memory Like This . . . .

The main memory capacity of 2 Megabytes on the Series III has grown 4X to 8 Megabytes on the Series 64 and 68.



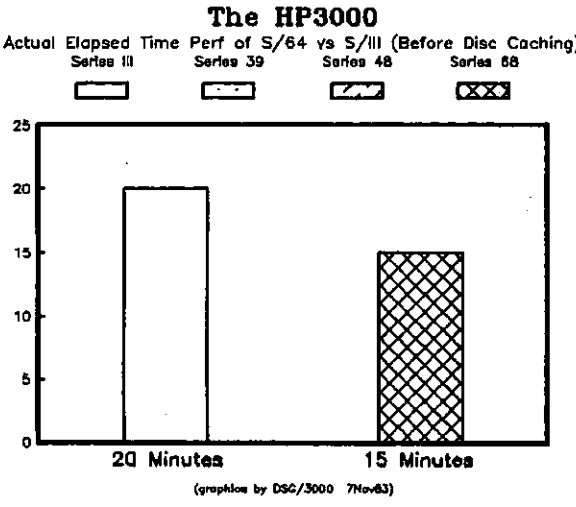
. . . You Get a Power Curve Like This . . . .

When you put these components together you get some entropy (like CPU consumption from the overhead of moving more I/O load) but a lot of synergy (like more supported terminals to utilize the greater disc capacity) so that your data processing "power" looks something like at least a 4X gain.



. . . So We Expected This . . . .

With 4X the data processing power, we thought we could take 20 minutes' worth of work on the Series III and do it in 5 minutes on the Series 64.



. . . But We Got This . . . .

When we took the 20 minutes of work from the Series III and ported it very carefully, 1-for-1, to the 64 and ran it there, we found we got only about a 25% improvement in performance.

The Bottom Line

Somehow *it wasn't getting there.*

If It Hasn't Got It There, It Hasn't Got It

We knew from our unit tests of individual components that we had the raw horsepower in the box to do better than that. But in the systems test, benchmarking customer data and customer programs, it wasn't making it to the Bottom Line.

The Autopsy of Our Shattered Dream

We started digging into the problem. We looked at some of the stats from the instrumentation we usually enable when running a benchmark. We found that we had the *CPU busy less than 50%* of the time and paused for I/O over 40% of the time. We were idling at a 75 nanosecond rate -- a very *fast idle*:

We looked at main memory utilization. Of the 8 Mbytes we had configured we were using between 5 and 6 Mbytes, *with 2 Mbytes standing idle.*

We looked at disc I/O. We were doing *only 50 I/O's a second* on a system where our unit tests had sustained better than *3 times that load.*

We found that we were getting a *process stop about 40 times a second from impedes on the database control block* (single-threaded to assure data integrity and correct linkage.)

For a number of reasons, the majority of which were directly or indirectly related to disc I/O, we were not realizing the performance potential of the Series 64 when we did a simplistic migration of the typical HP3000 job mix.

The Birth of Disc Caching

Enter the Busch-Kondoff Team

About that time, John Busch, architect of the MPE Kernel, and Al Kondoff, a disc I/O specialist in the MPE lab, focused on the problem.

The Memory Hierarchy

They looked at the time it takes to get data from main memory. That's about 400 nanoseconds. They looked at the time it takes to get data from disc. That's about 40 milliseconds. *A 10000X difference.*

The Typical HP3000 I/O Profile

They did extensive sampling of a wide range of HP3000 job mixes. They built instrumentation to measure and characterize fetch sizes, temporal and spatial locality, read:write ratios.

They built an I/O simulator and analytic model to evaluate design alternatives and emerged with a new idea.

The Mechanics

John and Al devised a scheme that would use the 1-2 Mbytes of idle main memory on the 64 to store large blocks of disc data that were local to data specifically requested by the user. If you asked for record #1, they'd read in record #1 and the next 20 Kbytes or so of data, on the assumption that you'd ask for record #2 soon and other records in that vicinity. As these disc domains get cached into main memory, when you asked for record #2, you'd get it in *4 milliseconds rather than 40 milliseconds. That's 10 times faster.*

From the analysis of real-world I/O traffic patterns, it looked as though you would get a hit in the disc data cached into memory about 70% of the time - more than enough to make the several millisecond disc cache overhead a high-yield investment.

Not a Pyrrhic Victory

The Cache management was integrated at the Kernel level so that disc cache was competing for main memory via the same approximate LRU (Least Recently Used) algorithm that manages code, stack and extra data segments. This enables a dynamic sensitivity to the varying resource needs of the system at the global level. That way you don't sacrifice some other important sector of performance just to peak your disc I/O's.

When John told us about his simulation and analytic model results, we were very interested. It sounded as though he were on to something. Something that we really needed in the Systems Performance Center.

The Systems Performance Center

In Cupertino, California, the location of the key HP3000 R&D Lab, the Systems Performance Center runs customer benchmarks and does performance characterization of HP3000 hardware and software products.

A typical scenario of the Performance Center operation goes like this: A customer has a Series III and has outgrown it. He wants to see what response time and transaction throughput rates he would get with a 64. He and his SE and Performance Specialist in the field analyze his job mix and select programs and data for a benchmark. (They usually expand the load in order to do a little *ad hoc* capacity planning along with the benchmark.)

The field/user team then makes reservations in Cupertino and joins the System Performance Team there to run the customer's programs and data on equipment in the Performance Center configured to reflect what has been proposed as a solution.

The key element in this exercise is its *real-world connection*: it's customer programs running customer transactions against customer databases, in many cases, ported directly from the customer's production environment.

On-Line Dimension

A major contribution of the 3000 is to bring the on-line interactive dimension to the world of commercial data processing. With the customer still on the phone you can see that you have a hundred widgets in stock that you will ship him today, and 500 more are coming out of QA tomorrow that you will Federal Express to him. He says "Go", you update the inventory (so nobody else gets promised the same 100 widgets) and launch the order. Now. *In almost real-time.*

This is the world of the HP 3000. And this is what customers want to benchmark. But with the Series 6X, they want to see the response time they will get with a hundred terminals going at the same time. And what will happen to response time if they add a Meg of Memory? And what will happen to response time if they add another 7935 disc drive? Another GIC? Another ....

Benchmarking Tools

Well, how do you get a hundred terminals, hook them up, get a hundred operators, get them to enter data with the right think time and typing speeds, make errors only when you want them to, measure the response time at each terminal for each transaction (down to a hundredth of a second), get them all to

start at a specific time, stop at a specific time, not sneeze or stop for coffee except when you want them to, then do the whole thing over again exactly the same way in every detail after you have changed the disc configuration for the next test (which will be about midnight, at which time you want all hundred operators and timers and terminals and files on the mark, ready to go -- and it's Saturday night)?

How?

No how.

You can't do it.

Except maybe with a computer. Like an HP 3000, for instance.

TEPE

You take a terminal emulator like HP's TEPE (Terminal Emulator and Performance Evaluator) and write "scripts" for it. You tell the program the think times, typing speeds, baud rates, programs to run, answers to give to the program you're running. You build a hundred of these, varying whatever you want to vary, then compile them into a form that the computer can use.

Then you take a couple of 44's and configure them with a special terminal driver. You get some special TEPE cables that hook right onto the ADCC (terminal handler) edge connector on one end and onto the 3-pin RS232 ATP connector on the other end at the Series 64 junction panel.

The System Under Test

On the 64 (the System Under Test) you load your application programs and databases and other files. On the 44's you start up the TEPE monitor, tell it what scripts you want to run with what start times.

Then you launch. Each 44 takes 50 scripts, analyzes the parameters and sends an RS232 bit serial message down the wire to the 64. By the time they hit the 100 ATP connectors on the 64 junction panel it looks just like what would have arrived if you and I and the other 98 operators had been sitting there banging real terminals. Only it's *exactly repeatable. And controllable. And changeable.* And response time is measurable at a fraction of a second. And it's *economically and ergonomically feasible.*

Emulation

This is *emulation. Not simulation.* It's real RS232 messages coming over real wires to a real System Under Test. This is how you execute and measure large configurations of on-line systems.

While you're running the benchmarks, the TEPE monitor on the driver 44's is measuring the inquiry-to-first-response response time on each "terminal" so at the end of the benchmark you can get mean resonse time, min, max and standard deviation for individual terminals or for the whole load, for the entire test or for a selected steady-state window.

Over on the System Under Test, you enable instrumentation like OPT (On-Line Performance Tool), to get CPU, main memory and disc I/O statistics with variable granularity: global total, global by interval, logical device level. You can get system table utilization, working sets by process, and other components of a wide range of performance statistics.

Disc Caching Into Battle

It was into this arena that Disc Caching came to prove itself. This was *real-world programs, real-world data, real-world loadings, real-word response time requirements, right out of real-world customer production systems* around the real world. Where it deviated from the real-world was where customers injected worst cases (eg: zero think time) to stress-test the system.

Now remember where we performance engineers were coming from: we'd examined John's and Al's design. They had done their homework. Their design was esthetically beautiful. Their implementation showed the consummate art of skilled craftsmen. And we wanted very much to have something that would enable the 64 to do the job it was built to do.

But in the Performance Center *the real-world Bottom Line is what counts. If it hasn't got it there, it hasn't got it.*

The First Round

We went to the Benchmark Library to get some of the tougher customer tests we had run. We selected one with a 56 data-set IMAGE database and a couple of KSAM files. We set up for 21 terminals and gave it a shot. We first ran with a pre-Cache Q MIT (the official current MIT at that point in time) to calibrate a base line. One of the design criteria John and Al had set for themselves was not to degrade performance on a system where Disc Caching was not enabled.

We then restored the database, reset the system, updated to the Cached Q-MIT and ran the same 21-terminal test with Caching disabled. We compared the numbers (overall mean response time inquiry-to-first response) and other stats of finer granularity and found no degradation due to Caching (in fact, there was a small improvement from some minor optimization John and Al had done while they had MPE "open on the table.")

The Caching Parallel: Some Gain

We then re-staged (reloaded the databases, etc), enabled Caching on all LDEV's and repeated the 21 terminal-test. We got about a 20% performance improvement. That was better than nothing, but like tapioca -- *good, but not exiting.*

As with the other tests we were running with OPT and MPEDCP (MPE Data Collection Program) enabled with

logging at 60-second intervals. This, in addition to the TEPE response time stats as seen at each "terminal", gave us a wide range of granularity in monitoring this extensive instrumentation.

John and Al pored over the stats, and proceeded to optimize the code. We staged for the next shot and they gave us a new build. We ran with a 25% gain this time.

The queue length at the LDEV level showed that we didn't have enough load with only 21 terminals to get the full "Cache Effect." We went to 42 terminals, then to 63 and started to get 50% (2:1 response time reduction) improvement and better.

Stage, Shoot, Tune, Build

We worked into Saturday night with a Stage-Shoot-Tune-Build routine, tweaking things like the cache-flush algorithm and fetch size. Most iterations picked up a few points. Some were disasters that we had to undo. We were tired but encouraged.

We came back Sunday and went into the Stage-Shoot-Tune-Build mode again. We pulled in some other benchmarks from the library. We ran some sequential file processing. With the larger fetch sizes and higher cache hit rates we got some very impressive results.

As the week wore on, we widened the range of benchmarks. We took the 56-dataset IMAGE benchmark with 63 terminals doing transaction processing and added 38 developmental (COBOL, FORTRAN, BASIC edit, compile, test) sessions. Here we had a hybrid benchmark with 101 terminals accessing IMAGE, KSAM, and sequential files.

We ran with Caching disabled. Mean Response Time (Inquiry to First Response): 0.92 seconds.

3:1 Improvement

We restaged, enabled Caching and ran again. Mean Response Time: 0.31 seconds. A 3:1 improvement. We knew we were on to something big.

After two weeks we cleaned up the debris, sifted through our data, had an engineering review by the development team and the full Performance Engineering team. We decided to integrate a Disc Caching parallel phase with every benchmark coming through the Performance Center.

Looking for the Holes

We were getting very impressive results where we had a good fit:

20-40% available CPU
1-2 Mb available main memory
Average file locality on disc
Average (3-4:1) Read:Write ratio
Heavy disc I/O load

We had found that there were conditions where Disc Caching not only didn't help but actually degraded performance. We focused on defining those conditions, so as to minimize the occurrence of misfits.

We found these conditions where Disc Caching had marginal or negative effect:

CPU busy >65% of the time
Memory Pressure >10%
Low hit ratio in a 4 Kb IMAGE or 24 Kb sequential fetch size
Block size about equal to Caching fetch size

Very Good Becomes Better

As we gave Disc Caching more and more real-world exposure John and Al added some features to make Caching more flexible.

In addition to the STARTCACHE & STOPCACHE MPE commands at the LDEV level, they built a CACHECONTROL facility that would allow user specification of fetch sizes for both sequential and random files. This enables some remedy to situations with a locality problem. The fetch size can be expanded to get better hit rates; or, if that doesn't remedy the problem, the fetch size can be reduced to avoid the overhead of futile Caching.

For IMAGE integrity-sensitive applications where recovery is difficult or impossible, John and Al built a CACHECONTROL BLOCKONWRITE facility. This enables a user to force a process wait until the completion of physically posting the cache buffers to disc. This works at the systemwide level.

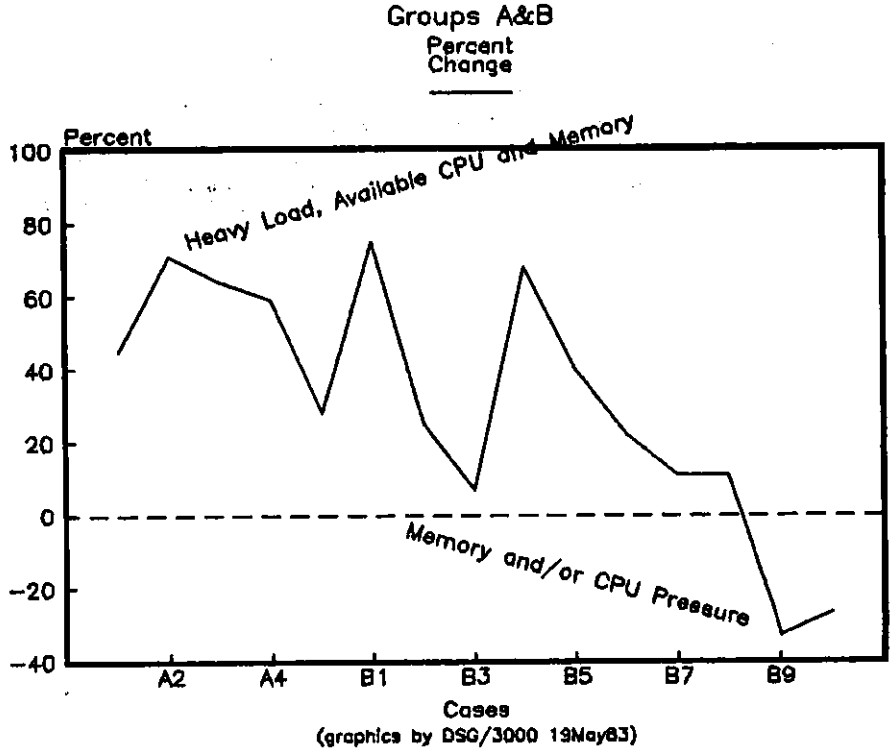
Alternatively, if IMAGE Logging and/or ILR (Intrinsic Level Recovery) are enabled by the database manager, the Serial Write Queue facility protects the integrity of the database.

A Performance Profile at Product Announcement

By the time Disc Caching was announced in May of 83, the Performance Center had benchmarked the product for four months. Just prior to the announcement a preview edition of the Performance News Notes (the monthly publication of the Performance Center) went to the field SE Organization with performance data like the Improvement Rate chart below.

This issue also had an internals description by John Busch and Al Kondoff. The combination of theoretical insight coupled with real-world performance data over a wide range of conditions gave the field SE's a head start in learning the characteristics of an important new product and the conditions that make it a good investment for the HP3000 user community.

Disc Cache: Improvement Rates



About the Authors....

Becky McBride *is a Performance Specialist in the Systems Performance Center at the Hewlett-Packard computer facility in Cupertino, California. Becky's responsibilities include performance benchmarking and product characterization, with special emphasis on HP3000 application products. Becky joined HP in 1980, after receiving degrees in Computer Science and Accounting from Ohio State University.*

Sam Boles *is a Systems Engineer in the Systems Performance Center at the Hewlett-Packard computer facility in Cupertino, California. With HP for seven years, Sam's computer experience started back in the AUTOCODER days of the 1401/1410, migrated thru the 360/370 era, and now focuses on HP hardware and software performance characterization. Sam earned his MS at UCLA in Information Systems.*

sebiug47 0215 15Nov83