Question
Isaac Aaron · Jun 3, 2016

Read I/O Performance On >50,000 IOPS

Hello

During some consultant activity I did at a client's I have discovered something very interesting. It seems like the current processing cycle as written in ObjectScript has trouble utilizing an SSD-based storage machine with five-digit IOPS.I have read some the articles including the one by Tony Pepper here, which is good for reference.

The article by Tony Pepper discusses "Random Read I/o Storage", which was exactly what I was testing. By looking at the "Specifications and Targets" section I can tell immediately what the difference is between performance as perceived by this article (and the others I've read so far) and the challenges we face on a highly capable machine such as the IBM V9000 storage or Hitachi Data Systems G200 (an up) configured with SSDs.

On Tony Pepper's article, the storage system is a 24-disk machine with 10K RPM disks. These machines can generate about 2880 host IOPS. On these conditions Cache excels in performance because disk seek times are considerably slower than the computer's RAM and the CPU's capability to span through data received from the storage machine.

An IT process usually written by a programmer with medium skills benefits greatly from being able to run a $ORDER based loop on a global while performing the usual span of read, process, write tasks. Cache also takes into consideration that write operations are more expensive than read operation by implementing write bursts with the write daemon. Compared to a typical SQL server client cursor process, Cache eliminates the network bottleneck and latency caused by having to exchange data with the client all through the process as it reads data packets, processes and performs atomic writes for each record.

The big difference comes when the storage system is much more capable than 2880 IOPS. New SSD based systems provide typically 60,000 IOPS and I have found that to be a game changer when trying to apply the same work practice as mentioned before ($O, read, process, write). In my test I took a 160GB global and wrote a simple process that starts at a random point from that global, spans forward with a $O run, advancing by about 200 records each 10 records read (to force a new seek out of the current block), reads the node at the current iterator and optionally writes the same data on it. My aim was to squeeze as much as IOPS I can get from the storage controller. I ran multiple jobs of that process experimenting with the numbers by changing the number of read and R/W processes.

The outcome was quite surprising. At first I had some disappointing results and I couldn't see any resource used up its maximum - neither storage nor host. So I then increased the number of jobs and then had to face some challenges keeping the virtual machine responsive. I later discovered that the problem was a break of balance cause by the write daemon waking up for its delayed write which requires CPU while the machine's resources were almost exhausted after I kept increasing load to its maximum capacity.

I then realized that the real issue here is with reads. One ObjectScript Interpeted process on the test machine doing a simple $ORDER run with a simple read (Set X=^GLOBAL(SUBSCRIPT...)) could only employ the storage machine for a few hundreds of IOPS. The CPU overhead involved in executing the ObjectScript code makes the read ineffective. Each line I optimized raised the IOPS count. The best I got with the test machine was about 8000 IOPS. On that same machine, running an Integrity Check utilizes about 40,000 IOPS and I could never reach that high with ObjectScript code.

Now back to the SQL Server scenario. In SQL Server - what the client usually does is ask for a result set. By asking for a result set it hints the SQL server on how the full data criteria is going to be, and then the SQL server can run its highest possible efficient code to obtain the data in read-ahead chunks. The practice is fairly efficient because the client process does mean to read through all that data and this is not just a statistical mechanism

assuming that the client is going to want the next/adjacent block of data. The client-server protocol is also designed to read chunks of data, reducing the network overhead. Writes can also be aggregated by using server-specific techniques (certain kinds of transactions, delayed writes).

What I'm looking for is a similar mechanism where I can hint the server on the required result set and expect a CPU-efficient process to manage the task. ObjectScript will function differently depending on how strong the CPU is, but I estimate that with something considered as a high-end server today is going to be slow on 20,000+ IOPS.

This worries me when I think about the many programmers I know who maintain code migrated from [IMD]SM and still code in ObjectScript. On heavy load processes, their programs usually do nested $ORDER loops doing random reads, processing and writing. For them, usually a storage upgrade used to be salvation. Now it seems like these processes some sort of a limitation that's going to be hard to defeat: random reads on an efficient storage controller requires a handful of CPU time and there's still the CPU time required to do that processing the program wanted to do to the data in the first place. So we might be seeing some weird cache servers consuming 100% CPU while the reason for that might no be inefficient code, but the exact opposite - very efficient code with minimum lines between READ commands.

There could of course be a different direction I have not thought of or I could be running the tests wrong and I will of course appreciate any input.

One final note: Cache *is* super fast. Those SSD/60,000IOPS high end machines are at this point cutting edge technologies for very specific customers. What concerns me now is how InterSystems is going to handle that kind of hardware if/when it becomes a commodity. I'm quite sure Cache is going to keep being super fast when that future is to become present. Just as it handles machines with 24+ CPUs today when it used to have issues with it in the past.

#Caché #Field Tests