From: Spitzer, Andras ([redacted])
Sent: Friday, April 09, 2010 10:34 AM
To: [redacted]
Cc: [redacted]
Subject: RE: I/O profiling on cseloukpuapp13

Anders, Damian,

Here are my findings of this week. I ran and wrote new stats/diags to measure/monitor any possible bottlenecks, **I've been monitoring the CPU, memory, and I/O subsystems**, on the physical and also on the logical layer. What I found is while I was mapping the pattern how this batch program works it doesn't really have one substantial bottleneck, but it seems like that's the best it can do in this environment. What I mean. For example you can see that the process never goes above 10-11% CPU usage, you might say that it means that the CPU isn't the bottleneck, then I diagnosed the I/O subsystem where it proved that the I/O is not a bottleneck by itself either (I disabled the backup which was generating a huge amount of I/O on Sunday, and the batch job finished late again), so the question is why the CPU usage can't go beyond 10-11%? I monitored the CPU scheduler this week, and it turned out that the process is sleeping a lot, waitig for I/O, which is normal as most of the syscalls are I/O syscalls. This means that when the program runs, it's using 100% CPU obviously, although as it happens only for peaks, the stats which are measured in a period of time will show us only 10-11%. Stats are really bad in showing peaks.

After all these tests I didn't want to waste more time chasing a ghost, so **instead of "why it's slow(er)" I switched my concept of "how can we make it faster".**

Anyway, **here comes the facts I found** about this batch program :

- the **majority of the system calls are fcntl() (file locking) and pread()** (reading from offset)
- **most of the process time is spent executing pread()** (which indicates that this is a read heavy application)
- a problem is that **99% of the pread()s are 512byte reads**, this is an unfortunate design
- CPU usage doesn't go beyond 10-11% because it's **an I/O heavy application** which makes the LWP to go sleep/wakeup constantly
- the program is a **single threaded** application, as such it has only one LWP, which makes it impossible to benefint from multiprocessing
- most **heavily used LUN** during this batch program **: ssd9** : c4t500604844A378D67d137/c5t500604844A378D68d137
- program is **using only ~20Mbyte memory**, which is really not much
- the amount of I/O the program is **reading/writing not that much**, how it reads/writes makes it less efficient (512byte pread()s, this makes 100kbyte to read ~400 syscalls (~200 fcntl() and ~200 pread()s)

- I analyzed the filesystem, **vxfs filesystem and it's unfragmented**
- I analyzed **the vxfs buffers and caches,** everything looks **normal**

- during last night I analyzed one additional logical bottlenect, it's a deep kernel parameter to limit the outstanding SCSI commands per device (I assumed we might have reached that limit),the batch program is not affected by the limit at all

This was the OS analysis, and as unfortunately I couldn't really spot one specific bottleneck, I went further and I started to look from **the application standpoint**. The batch program we talk about us a Cobol program, using AcuCobol-GT 7.2.1 runtime environment, and as I have no access to the source code, I was trying to find parameters which could make the run time more efficient based on my profiling I did earlier in the OS. The batch command executes as the following : "runcbl -c /appl/etc/640_config c01p60

1". Runcbl is the AcuCobol runtime environment, /appl/etc/640_config is the config file, c01p60 is the Cobol object file's name (c01p60.run), and '1' is a parameter passed to the program.

If I have had the source code, I could have diagnosed the type of transactions we are using, so I could have suggest application specific parameters to tune. As **I had no source code access, I make my suggestions based on the I/O/CPU profiling** I did with Dtrace from the OS.

Here are **my findings about the Cobol environment** :

- the batch job is **compiled with debug enabled**, this obviously causes some overhead which makes the performance less efficient :

```
cseloukpuapp13% /appl/acucobol/bin/cblutil -info
/appl/backup/fabpgm/c01p60.run
/appl/backup/fabpgm/c01p60.run:  size: 25726 (647E), v5.1, VAX-mode, tables
checked, debug (19-Mar-2010 13:52:49)
```

From the docs : "In each report, cblutil includes information that indicates whether the module is in debug-mode. Because programs compiled for source-level debugging can be quite large, it can be helpful to run reports on a regular basis to see if you have accidentally left any programs in debug-mode." This indicates that for a program in production the debug might be disabled.

Recommendations to the AcuCobol environment :

- first of all, I recommend to **enable profiling** on this particular batch job. With the help of profiling the Cobol Runtime environment will collect stats about the batch job, how much time is spent on I/O, CPU, etc. To do that, there are two requirements : **1#** the program must be compiled with debug (luckily? We have this one already) **2#** we have to add a '-p' to the command line of the batch job. This means to add the '-p' flag to the shell script : /appl/bin/start_c160_640.sh

After we add the '-p' flag to the command, during next run it will collect stats (and create a file acumon.xml which can be further processed with acuprof) about the execution and we'll have some ideas where to look next. (at the end of this troubleshooting of course I recommend to recompile the batch job without debug enabled, unless you have a reason to have it enabled)

- **irrespectively of the profiling, I would recommend to tune the following parameters** in the config file :

- **SORT_DIR** : this is a parameter where the temporary files are stored during a SORT operation. By default this is the local directory, I recommend to set it to /tmp, as /tmp is a memory backed filesystem (tmpfs) which is way faster than the local disk.

From the docs : "2.7.5.1 Sort files

The SORT verb often makes use of temporary files. By default these files are stored in the current directory. You can specify an alternate directory to hold the sort files by setting the configuration variable SORT_DIR to the desired directory. This value is treated as a prefix, so any trailing directory syntax (such as "/") is required. You can improve the performance of the SORT verb by placing the temporary files on a fast device. Care should be taken, however, that the device has enough free space to hold twice the size of the data to be sorted. "

- **SORT_FILES** -> defaults to 4 (4 to 32) : I would recommend to raise this value, maybe to 8 (doc states that raising the number of sort files can result improved performance, the only limitation is the number of open filedescriptors, and I doubt we've reached that)

- **SORT_MEMORY** -> defaults to 4 (=256Kbyte), n x 64kbyte to allocate (1 - 4096) : currently the sort memory is using the default setting which provides the Cobol runtime environment 256Kbyte buffer for sorts. This is not much, I would recommend to raise it to at least 600 (which would give 38Mbyte sort buffer)

- **V_BUFFERS** -> defaults to 64 (32kbyte !!), n x 512 byte to allocate for indexed files (0 to 2097152) : as this application is using database files in 95% of it's time, I would recommend to raise the standard buffer for indexed database files, which is defaults to 32kbyte. I would recommend to raise this to at least 192000, which would result of having ~96MByte buffer. This parameter makes sense only if you are using Vision files, but I assume you are using Vision files (more precisely you are using Vision files version 3).

As you can see the defaults of these parameters are really designed for low capacity servers, and as your system usually have at least ~4-5G memory available, you could raise these values even higher, and I hope it would provide a better performance for your application. I will keep on analyzing the system/application, if you would enable profiling for your batch job, it would help us all to diagnose where to look next to improve performance. Also, please consider amending these values I mentioned above, just read the docs to confirm it's a good idea to tune these, because I really believe the defaults were sized long time ago. (that's why the batch job is using ~20Mbyte only)

Also, please find my I/O profiling sheet attached, this version also includes my stats about the system calls (just see the sheet 'Graphs - April 06'), it can show you what I wrote earlier about pread() and fcntl(). Also for the record, I made my suggestions here based on the AcuCobol-GT 7.0, and you are using 7.2.1, although I assume these should work also at 7.2.1.

Regards,
sendai