Lecture 29 —Profiling and Scalability

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

November 13, 2020

Goal: scale number of users,

$$1 \rightarrow 100 \rightarrow 10000000.$$

Be more fast?
You have other options too…

"In the midnight hour, she cries 'more, more, more"' - Billy Idol

Don’t Guess, Measure

As always, we want to measure (profile).

3 principles to think about
when going from dev to production:

Hardware Principle

Scalability testing $\neq$ QA testing.

Dev + QA on local computer.
Your concerns: Is it right? Is it fast enough?

Fine, but it's no way to test if it scales.

Hardware Principle

Test on prod-like machines.

Low-end systems have very different limiting factors.

Your laptop: limited by 16GB of RAM.
Prod server: 128GB of RAM.

So you might spend a great deal of time worrying about RAM usage and it just doesn't matter.

Reality Principle

Brenan Keller
@brenankeller

A QA engineer walks into a bar.
Orders a beer. Orders 0 beers.
Orders 9999999999 beers.
Orders a lizard. Orders -1 beers.
Orders a ueicbksjdhd.

First real customer walks in and asks where the bathroom is. The bar bursts into flames, killing everyone.

1:21 PM · 30 Nov 18

Quality Assurance

Reality Principle

Use a “real” workload.
(As far as possible).

Maybe not actual live customer data,
but try to come close.

Limited test data/scenarios $\Rightarrow$
inaccurate test results.

Your tests run summary reports occasionally…
your users might run them every hour.

Volume Principle

“More is the new more.”

Lighter workloads OK for regression testing.

To see how your system performs under pressure, you actually need to put it under pressure.

Can simulate pressure by limiting RAM or
running something very CPU-intensive concurrently,
but it’s just not the same.

These tests also of great interest to the customers, who would like to know that you can deliver.

Reproducibility and Regression Testing

Science rule 1: results need to be reproducible!

Good programming practice: must run unit tests,
and re-run to make sure it all works.

Old (solved) performance issues also bad.

Or a new change that slows the whole program down still more bad.

Characterizing Problems

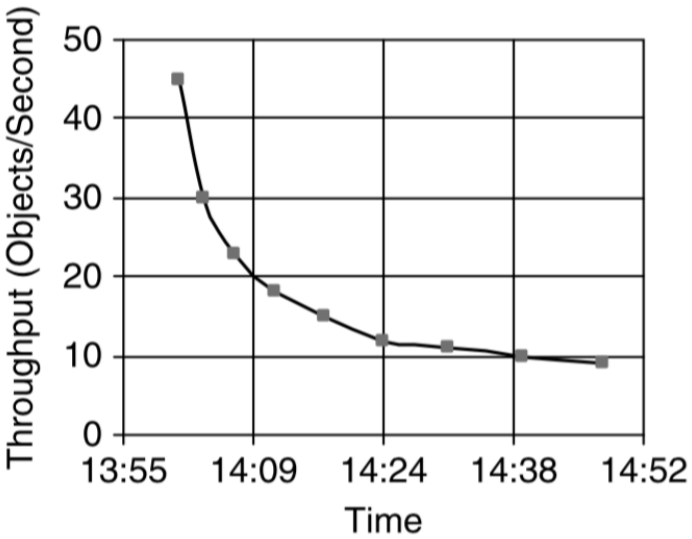
Example: application deployed on two systems—application server and database server.

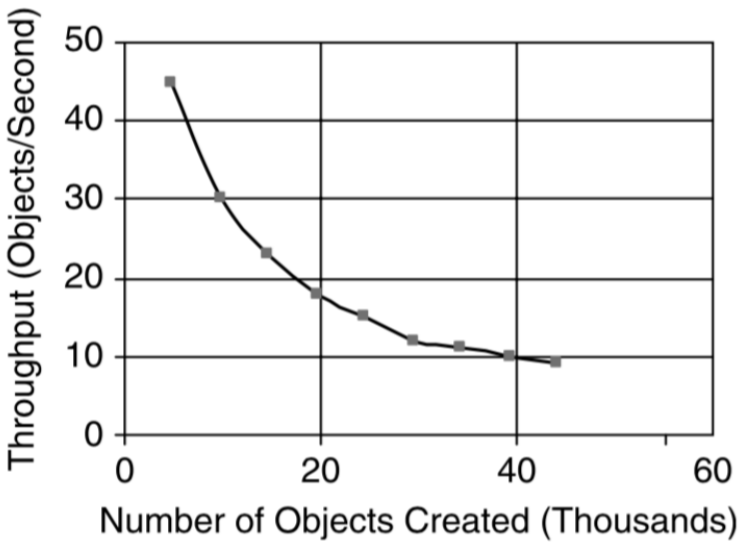
Data stored on an external SAN
with RAID0 configuration.

Java program simulated object creation
with 15 threads.

Performance metric: objects created per second.

Scalability Fail(ability)





This is, to use the technical term, “not good”.

Elementary, My Dear Watson!

It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.

— Sherlock Holmes
(*A Scandal in Bohemia*; Sir Arthur Conan Doyle)

Collect Evidence

Who's to blame?

1. CPU
2. Memory
3. Disk
4. Network

These are, obviously,
categories rather than specific causes.

Blame the CPU

CPU is probably the easiest of these to diagnose.

`htop`, Task Manager, etc. will tell you if CPU hosed.
Look at the %CPU columns
and see where all your CPU is going.

Overwatch

Still, that tells you about right now;
what about the long term average?

Checking with my machine “Loki”, that has since ascended to Valhalla:

```
top - 07:28:19 up 151 days, 23:38,  8 users,  load average: 0.87, 0.92, 0.91
```

Those last three numbers are the one, five, and fifteen minute averages of CPU load, respectively.

Lower numbers mean less CPU usage and a less busy machine.

Interpreting Load Numbers

Picture a single core of a CPU as a lane of traffic.

You are a bridge operator and so you need to monitor how many cars are waiting to cross that bridge.

If no cars are waiting, traffic is good and drivers are happy.

If there is a backup of cars, then there will be delays.

Single-CPU Load Scheme

1. 0.00 means no traffic.
   Anything between 0.00 and 0.99 means we're under capacity and there will be no delay.

2. 1.00 means we are exactly at capacity.
   Everything is okay, but if one more car shows up, there will be a delay.

3. Anything above 1.00 means there's a backup (delay).
   If we have 2.00 load, then the bridge is full and there's an equal number of cars waiting to get on the bridge.

Car Analogy

= load of 1.00

= load of 0.50

= load of 1.70

Is it a Problem?

$\geq$ 1.00 isn't necessarily bad, but you should be concerned if there is consistent load of 1.00 or above.

$<$ 1.00 but getting close to it: you know how much room you have to scale things up.

$>$ 0.70 then it's probably time to investigate.

$\geq$ 1.00 consistently we have a serious problem.

5.00: this is a red alert situation.

Now this is for a single CPU—if you have a load of 3.00 and a quad core CPU, this is okay.

Traffic analogy: four lanes of traffic, of which 3 are being used to capacity.

So we have a fourth lane free and it's as if we're at 75% utilization on a single CPU.

End Digression

Back to our example. Is it CPU?

The Application Server CPU utilization, on average,
was about 10% and on the database server, about 36%.

So that is probably not the cause.

Next on the list is memory.

Disk? Or Memory?

How to tell? Look at disk utilization.

Not enough RAM $\Rightarrow$ swapping, bad perf, no scalability.

(In the worst case.)

You can ask via `top` about how much swap is being used, but that's probably not the interesting value.

```
KiB Mem:   8167736 total,  6754408 used,  1413328 free,   172256 buffers
KiB Swap:  8378364 total,  1313972 used,  7064392 free.  2084336 cached Mem
```

Misleading Memory

Why? Memory being “full” does not necessarily mean anything bad.

It means the resource is being used to its maximum potential, yes, but there is no benefit to keeping a block of memory open for no reason. (Or stockpiling late days).

Also, memory is unlike CPU; if there’s nothing for the CPU to do, it will just idle (low power state).

Really, Dvorak, Really?

Memory won't "forget" data if it doesn't happen to be needed right now—data will hang around in memory until there is a reason to move or change it.

So freaking out about memory appearing as full is kind of like getting all in a knot about how "System Idle Process" is hammering the CPU…

Page Faults

You can also ask about page faults, with the command

```
ps -eo min_flt,maj_flt,cmd
```

Major page faults: had to fetch from disk.

Minor page faults: had to copy a page from another process.

The output of this is too big even for the notes.

This is process-lifetime data.

Swapping Report

What you really want is to ask Linux for a report on swapping:

```
jz@Loki:~$ vmstat 5
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0 1313972 1414600 172232 2084296    0    0     3    39    1    1 27  1 72  0  0
 0  0 1313972 1414476 172232 2084296    0    0     0    21  359  735 19  0 80  0  0
 0  0 1313972 1414656 172236 2084228    0    0     0   102  388  758 22  0 78  0  0
 4  0 1313972 1414592 172240 2084292    0    0     0    16  501  847 33  0 67  0  0
 0  0 1313972 1412028 172240 2084296    0    0     0     0  459  814 29  0 71  0  0
```

Swapping Report with Actual Swapping

```
  procs                          memory    swap          io     system cpu
r  b  w   swpd   free  buff cache  si  so   bi   bo   in    cs us  sy
. . .
1  0  0  13344   1444  1308 19692   0 168  129   42 1505   713 20  11
1  0  0  13856   1640  1308 18524  64 516  379  129 4341   646 24  34
3  0  0  13856   1084  1308 18316  56  64   14    0  320  1022 84   9
```

Skip to Disk

Looking at disk might seem slightly redundant if memory is not the limiting factor.

After all, if the data were in memory it would be unnecessary to go to disk in the first place.

Still, sometimes we can take a look at the disk and see if that is our bottleneck.

Looking at Disk Usage

```
jz@Loki:~$ iostat -dx /dev/sda 5
Linux 3.13.0-24-generic (Loki)  16-02-13  _x86_64_ (4 CPU)

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.24     2.78    0.45    2.40    11.60   154.98   116.91     0.17   61.07   11.57   70.27   4.70   1.34
```

Last column %util tells us what we want to know.

We can ask about the network with `nload`.

You get a nice little graph if there is anything to see.

But you'll get the summary, at least:

```
Curr: 3.32 kBit/s
Avg: 2.95 kBit/s
Min: 1.02 kBit/s
Max: 12.60 kBit/s
Ttl: 39.76 GByte
```

Back to the Original Question

The book contains the full story, which is maybe interesting to you if you wanted to dig into the specifics about Oracle 10g and SQL query syntax.

You probably don’t care about the details,
but 96.4% of the total database call time was attributed to database CPU

Top 5 Queries

Sure enough, the “top 5” queries were taking up a huge amount of time and they all look something like:

```
SELECT documentId, classId, dataGroupId, consistencyId
FROM objectTable
WHERE objectID = <value>;
```

Why does performance of this stink?

Top 5 Query Performance

Lots of reads from the database.

The reads themselves don't necessarily go to disk (cache/buffers/etc may save us here) but we're still doing a lot of reads of data.

To speed this up, what we need is to add an additional index for each of these tables.

This is one of the strategies we've talked about before—"be prepared".

The Impact of the Index