You don't Know Jack about Application Performance

Toronto Linux User Group, 12 Sept 2006

David Collier-Brown, Data Center Works
Everyone thinks performance has to do with resources

- Most programmers think if a program runs too slow, we should throw CPU at it, or memory or disk...

- But what about program that has all the CPU, memory and disk in the world, but still stubbornly refuses to deliver more than ten transactions per second?

- Should we perhaps find the bottleneck instead?
But it doesn't

- Resources are easy to measure, but customers don't care.
- Customers care about guaranteed low response times and lots of transactions per second, all at a low price.
- We need to measure performance first, then diagnose, and only if we have a resource problem throw resources at it.
- This is the old story of looking under the bright streetlight for the ring lost in the shadowy garden.
This talk is about

- What performance is, and why
- Measuring performance
- Programming for performance
- Benchmarking for performance
- Tuning for performance
What is Performance?

- Response to a load, in TPS or bytes/second
- Response in reasonable time.
  - Latency
  - Response time
- What's “reasonable” mean?
  - 1/10 second is fast
  - One second is not fast
  - Ten seconds is bad
  - Thirty is very bad
Why don't we measure it?

- It's hard
  - Or, optionally, brutally expensive

- Vendors could report it
  - They used to in the mainframe days
  - But they got screwed when they did

- So we make do with resources
  - And often we luck out, when there is a CPU or memory bottleneck
But things have changed

- Many applications use TCP/IP
- There are lots of packet capture tools to use
- There are also free benchmark tools (JMeter)
The Laws of Performance

- You may remember these diagrams from a textbook
- The operational laws dictate the shape of the throughput and response time curves
- They're only high-school algebra, but they led to queuing theory
The Queue (no theory involved)

- N users
- N requests/second
- S sec. service time
- W sec. Wait time
- R sec. response time
- D sec. demand
- Z sec. think time (hidden)

yields

- X transactions/second
The Throughput Curve

- The first curve is the upper bound on throughput ($X$),
- It rises with load until the program reaches 100% utilization and then levels off.
- Measured curves don't actually have sharp corners.

![Graph showing the Throughput Curve](image)

Figure 3. Throughput, expected and measured.
The Throughput Curve II

- If we measure the service time, $S$, we can use the...

- Utilization law, $U = X \cdot S$ where $U = B/T$

- Consider the case where $S = 0.10$ sec.
The Throughput Curve III

- If $S = 0.10$, 10 transactions will fit in one second
- 10 TPS is all we'll get
- If $S = 0.05$, 20 TPS is possible
- And 10 TPS is only 50%

Figure 3. Throughput, expected and measured.
You can't get utilization above 100%, because then 1/10 if a second would have to go into a second more than ten times.

This is the reason that the throughput curve isn't a straight line to infinity: it always rises with increasing load, but then levels off at 100% utilization.
We can compute the load that yields 100% utilization.

The user load at 100% utilization is called $N^*$ and is equal to $1/S$.

We computed it by setting $S$ to a tenth of a second, $U$ to 1 and solving the utilization-law equation for $X$. 
Why doesn't it have square corners?
Initially requests arrive independently, and don't interfere.
As we get closer and closer to 100% utilization, there's more and more likelihood that two will be requested at the same time, and the second will have to wait.
Queue Buildup

- Past 100% utilization, requests have to wait. In our example, the 11th request has to wait for the other 10 to complete.

- The queue length is computed from
  - Little's Law
  \[ Q = X \cdot R \]

- In our example, a load of 50 would yield a queue length of \( \frac{50}{10} = 5 \), and the average Response time would be \( (40 \times 0.1 + 0.1) = 4.1 \) seconds
In the queuing circuit in Figure 2, the queue is represented by the sequence of boxes to the left of S.

- The queue delay or wait time is \( W \), and the total response time under load is \( R \), the sum of \( W \) and the service time \( S \).
Response time is the second curve in Figure 1, which starts out fairly level and then rises as we approach and pass $N^*$. The slowdown is from all the waiting in queue.
Response Time II

- If we did a benchmark, the response time would
  - start off horizontal, just like our diagram's initial line
  - then start to drift upwards fairly quickly towards paralleling the second.

![Graph](image)

*Figure 4. Response time, expected and measured.*
Benchmarkers "know" response time grows gently and linearly because their benchmark from 0 to 10 requests per second was relatively linear.

They never tried at 20 requests/second!
Consider the two response times that we mentioned before, one second and ten.

The proper equation predicts we'll hit the ten second mark at 107 requests per second.

The bad/linear equation would estimate we wouldn't hit the ten-second mark until 280 requests per second.

Only the customers (YorkU.CA) will know the real performance is less than half what they were promised. They and their lawyers.
Agenda

- What performance is, and why
- **Measuring performance**
- Programming for performance
- Benchmarking for performance
- Tuning for performance
Measuring Performance

- Many programs use TCP/IP, even locally
  - These you can measure with a packet capture
  - And there's a free benchmarking tool, Jmeter

- If it sends requests and receives responses, we can
  - measure the speed
  - predict both the curves
A Transaction Looks Like...

- At $t_0$, the request arrives
- At $t_1$, the first byte of the response is sent
- At $t_2$, the last byte is sent
- And we also record bytes transferred
We Measure

- Latency
  \[ t_1 - t_0 \]
- Response Time
  \[ t_2 - t_0 \]
- Transfer Time
  \[ t_2 - t_1 \]
- Throughput
  \[ \frac{\text{bytes}}{t_2 - t_1} \]
- Think Time
  \[ t_0 - t_2 \]
Response Time Looks Like

- Response time is latency plus transfer time
- This is a good sample, by construction
- Note the average, which we'll use in a moment
Throughput (Bytes/Second)

- The other kind of throughput
- Used for bulk transfers, like ftp
- Variable, but it didn't mess up the response time
We Can Compute

- *If and only if* we're below 100% and there's no queue
  - We ensured that when we measured it
  - The load was from wget with a sleep time
- TPS at 100% Utilization
  - Computed as 1/ Response Time
- Actual as a % of Maximum
- Queue length
  - By Little's Law, Q = XR
- And the the Slowdown due to Queuing
The expected line is $1/D_{\text{max}}$, which we arranged to be equal to $1/R$.

For one CPU, we're averaging $2.8/4.5 = 62\%$
Queue Length

- This is from one of the Operational laws, Little's Law, $Q = XR$
- Waiting in queue is what makes programs slow
● Throughput (X) = \min(1/D_{max}, N/D+Z)
● Response Time (R) = \max(D,D_{max}-Z)
● Where D is demand
● N is users
● Z is think time
● D_{max} is the largest demand
● And D \approx S \cdot 1
The Results We just Saw

- A few slow transactions
- We're at about 60% of capacity
- The queue length was about 0.6, and spiked to approximately 5.5
- At 7 TPS per CPU, we hit 100% utilization
  - At 28 requests/second, R will be 3 seconds, which is not what we want, but sort of ok
  - At 150 requests/second, R will be 15-30 seconds, which is bad
Why 3 and 30 Seconds?

- 1/10 second is fast
- One second is slow
  - Three seconds is the upper bound for slow with a watch cursor or some other apologetic message
- If the response time grows to 30 second, humans think the program is more than slow: they'll think it's crashed!
- 30 seconds happens to be the cache time of human short-term memory
Agenda

- What performance is, and why
- Measuring performance
- Programming for performance
- Benchmarking for performance
- Tuning for performance
Make performance part of your design

Build the performance test framework \textit{first}.

For example, the first day the code works...

Consider this test-directed performance design
If your target is 1/10 second, set your back end to almost that (maybe 0.09), and see if the front end gets in the way.

As soon as it's good enough, STOP. Don't waste your efforts making a fast part faster.

The maximum TPS will be set by the slowest part, and will be 1/Dmax, where D = S * Visit count. And visit count is number of calls to the slow part, such as a database or disk.
Tuning

● Your tuning in the front end will mostly be looking at code-path length with your framework and a profiler.

● The “HP” community is your resource here (High Performance as in Cray, not Hewlett-Packard)

● One reference is "Performance Optimization of Numerically Intensive Codes” by Stefan Goedecker (Society for Industrial and Applied Mathematics)
Then **Switch to tuning the SQL**

- Build a script that submits the SQL and measure it.
- Now you can tune the queries and the database structure.
- See “Optimizing Oracle Performance” by Cory Millsap (O'Reilly, 2003)
If you Have Middleware

- Arrange for it to communicate via sockets
  - It probably does anyway
- Measure it's performance the same way
- If you can't:
  - measure the front end and database
  - What's left is the middleware
And Now Look at Resources

- Find out how much CPU, memory and I/O each transaction takes at 1 TPS
- Now test up past 100% utilization, and see where it goes “haywire”

- Save that information for properly sizing your production system
- If you under-size a production system, you will introduce an artificial bottleneck
- That's what most “tuners” find and fix (and yes, that includes me)
Agenda

- What performance is, and why
- Measuring performance
- Programming for performance
- Benchmarking for performance
- Tuning for performance
Benchmarks for Performance

- In a TCP/IP world, benchmarking is easier
- First, check out JMeter
  - And Loadrunner, if you're rich or already have it
- If not, try
  - `wget -O /dev/null -w Z/2 –random-wait`
- Run for at least a minute at each load
- Don't just write down the results
Benchmarking Bugs Like to Hide

- Graph your results and look at the shapes
- Variations from the expected shapes identify the bugs
- Also, $X >> 1/D_{\text{max}}$, your load generator's lying (a common error)
Agenda

- What performance is, and why
- Measuring performance
- Programming for performance
- Benchmarking for performance
- Tuning for performance
Tuning for Performance

- The first thing is have enough CPUs
  - Not cpu speed, or % CPU, the *number* of CPUs
- Then look at latency versus transfer time
  - If removing either of these will make you fast enough, then you know where to look next
- Latency is sensitive to CPU and network speeds
  - But network bandwidth doesn't help here
- Transfer time is bandwidth-sensitive
  - Look at disk bandwidth first
  - Then at code length and code cost
  - Then look for resource starvation
Conclusions

- Start early
- Measure R
- Compute X at 100% utilization
- See how you're doing
- and finally
- Draw the graphs
Throughput

Response Time

Upper bounds of throughput = \( \min(1/D_{\text{max}}, N/(D+Z)) \)
Lower bounds of response time, \( R = \max(D, N \times D_{\text{max}} - Z) \)

To compute the throughput and response time curves, we start by measuring the response time at a very low load, so no queuing happens.

Response time, \( R \), at minimum load = 0.1

We now set the simulated users to issue 1 request per second, which allows think time, \( Z \), to be \( 1 - R = 0.9 \)

This only works if \( R \) is less than one second, so that \( Z \) is positive.

The maximum throughput will occur at \( N^* \), the load where \( N = 1/R \)

\( N^* = 10 \)

After \( N^* \), no improvement in throughput will be possible, and a queue will of work waiting to be processed will build up, causing the response time to grow without bound.

Our estimate for \( D \) and \( D_{\text{max}} \) is initially always \( 1 \times R \), so the variables are now all known.

\( D = 0.1 \)
\( D_{\text{max}} = 0.1 \)
\( Z = 0.9 \)