Predicting application performance

Mike Pagan
Principal Architect
HP Northeast Presales

Capacity planning vs. Tuning

- Tuning: the art and science of determining why a system is not performing well
  - Lots of tools: glance+, measureware, sar, vmstat, iostat…
  - Lots of books and best practices
  - Usually involves finding out that system usage has grown beyond the original deployment
  - In other words, tuning is usually troubleshooting!
- Capacity planning: the art and science of designing a system so that it meets users current and projected needs
  - Few tools
  - Few resources

"Tuning is common,

capacity planning is rare."

Me
HP

Why capacity planning?

- Cost: Deploy only what you need. No more, no less.
- Effort: Less sysadmin time backfilling and tuning an improperly sized system. Less sysadmin time and downtime upgrading.
- Adaptive Enterprise: In order for a UDC administrator to properly deploy a solution, the capacity necessary to support that solution must be known.
- Competitive Advantage: Most software providers can't really tell you which platform will provide the desired performance.

The capacity planning loop

Baseline

Benchmark: run only the benchmarks on a new or unknown system

Benchmark

Calculate: use pre-determined relations to predict performance

Baseline: determine the TRUE capacity of a known system by running the benchmarks and the application.

Calculate

Verify

Plan

Verify: periodically check predicted performance against actual and re-baseline if needed.

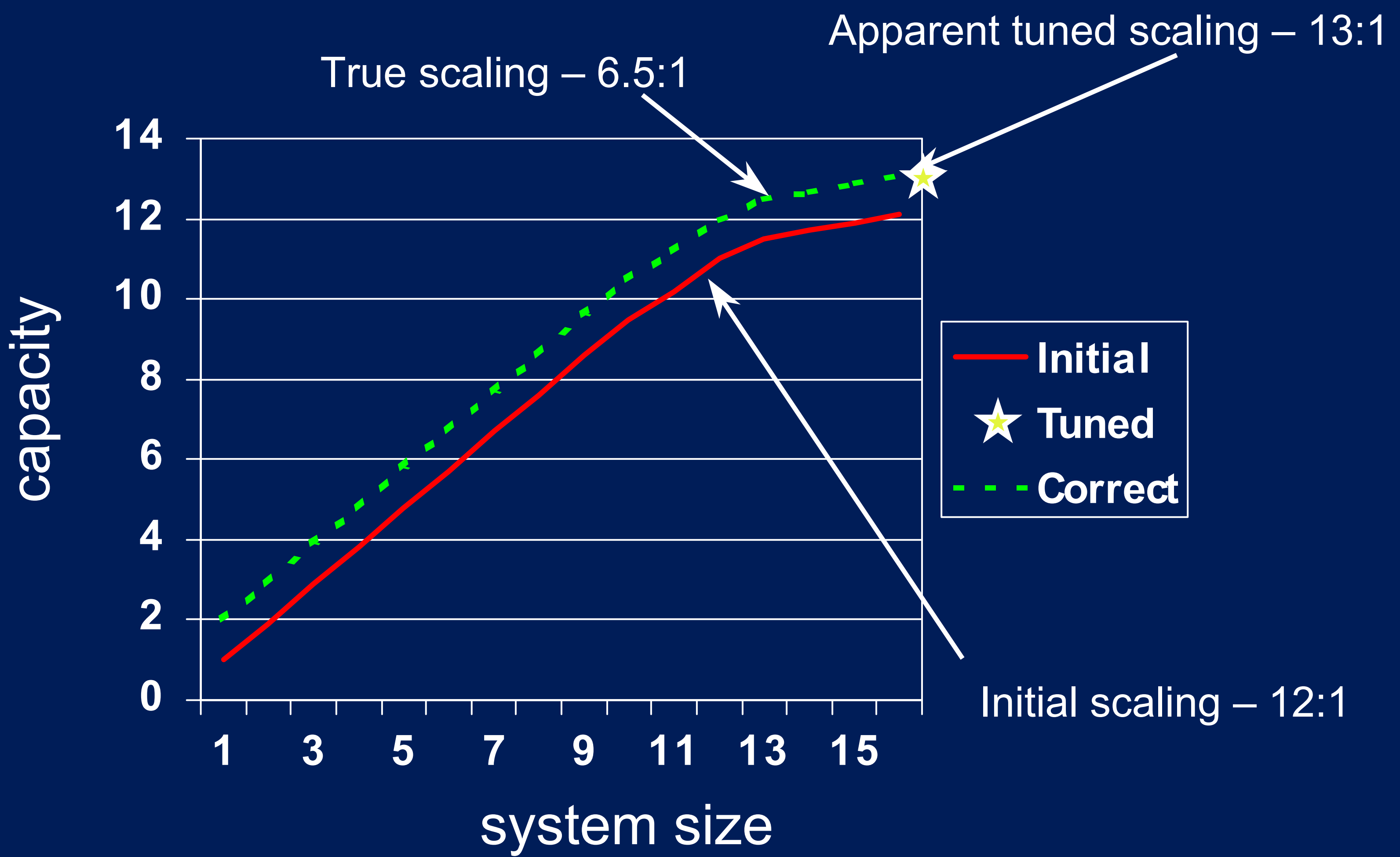
Benchmark: selection

- Selecting the benchmark is crucial to capacity planning
- May need more than one benchmark to properly model the application
- An ideal benchmark should be:
  - Specific, to match the application
  - Economical, able to run in reasonable amount of time with reasonable resources
  - Reproducible, i.e. insensitive to tweaks and tunes
  - Standard benchmarks are preferred (because other people run them for you)
- Of course, you'll never find the ideal benchmark so you must make do with what is available!

Benchmarks: The problem with standards

- Although standard benchmarks (Spec, TPC) are preferred, they also pose problems
- Manufacturers published results can cause trouble due to sparse coverage and overtuning
  - Sparse coverage: only running the benchmark for certain configurations of hardware and software
  - Overtuning: setting parameters that improve benchmark performance but which don’t reflect real-world practices
- But of course HP would never do such things ☺
- You can still used standard benchmarks, just know their limitations

Benchmarks: problematical published results



Benchmarks: selection & types

- Synthetic benchmark
  - TPC-C, SDET, Volano, Linpack
- Partially synthetic benchmark
  - SpecINT, SpecFP, Ariba, SAP
- Natural benchmark
  - Custom

- Examine your application & know it's characteristics in order to match up with a benchmark
  - Language: Java? C++?
  - Middleware: Oracle? SAP? BEA?
  - Behavior: single threaded? multi threaded? Disk intensive?

Benchmarks: selection & types cont'd.

- Benchmarks should be reasonably cost effective
  - Run it in a few hours or less, not days
  - Set it up in a day or less, not weeks
  - Baseline it on hardware that's affordable, not a Superdome/128 with 60 Terabytes of high-end mass storage
- TPC-C fails for this purpose because it's too costly, too complicated, too high-stakes (published results)
- TPC-C is still good for other purposes (comparing platforms prior to creating or deploying apps)

Benchmarks: selection & types cont'd.

- Benchmarks must also be able to run in the available time
- Example: scalability benchmark
  - If I want to run it with 5 different CPU configs (1,2,4,8,16)
  - …and 5 different disk configs (5, 10, 15, 20, 25 spindles)
  - Then I have to run the benchmark up to 25 times
  - If each run takes 2 hours, that's over a week in the CPC

Benchmarks: running the benchmarks

- Above all, to thine own self be true
  - In other words, you run the benchmarks so you get to run it in good faith for your own purposes. Cheating doesn't help you.
- You still need to tune the system properly, but it does you no good to "overtune" the system and squeeze every last CPU cycle out of it
- Eliminate extraneous bottlenecks and benchmark only for the key parameter (usually application throughput)

Benchmarks: parameters to “tune out”

- RAM
  - Should have an excess of physical memory
  - No swapping or paging
  - If you don’t, you’re only testing the virtual memory subsystem
- LAN
  - Look at pps, collisions, throughput: none should be more than 80% of max
- Disk I/O
  - This may not be a parameter to “tune out” for disk intensive apps
  - If app is disk intensive, use a CPU throughput benchmark and a Disk I/O benchmark seperately (Teamquest)

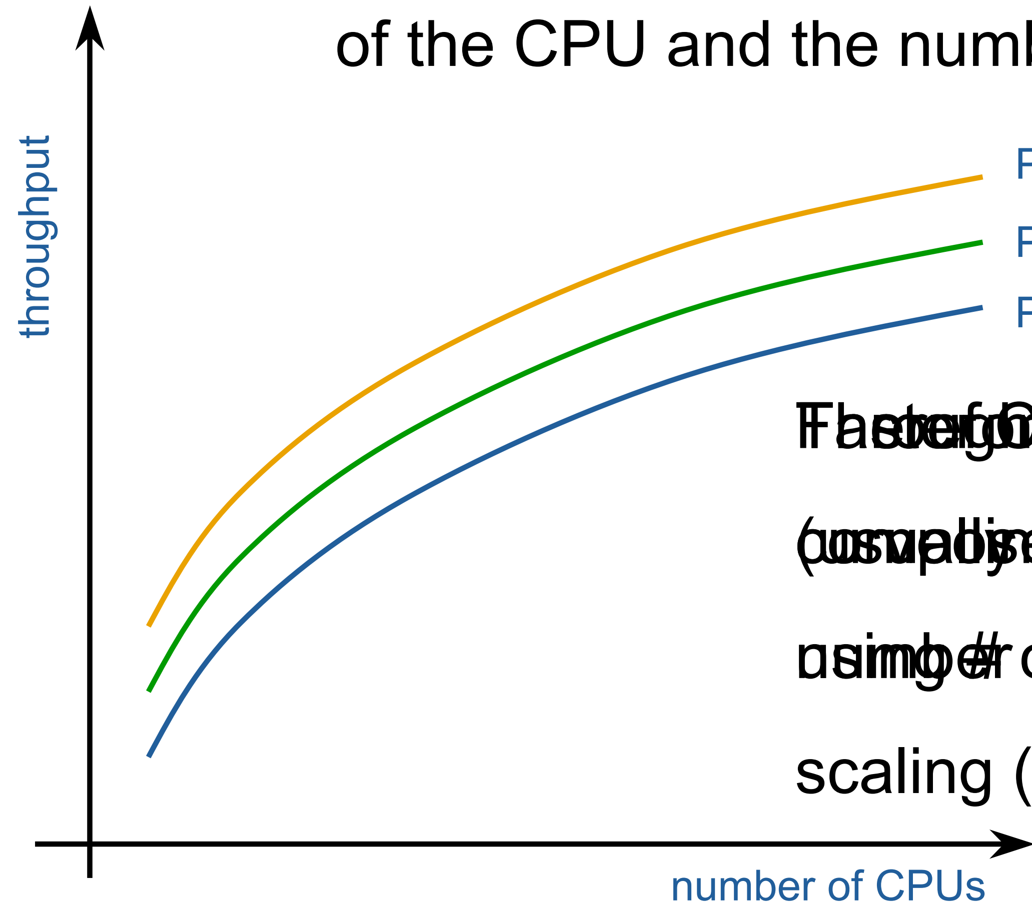
Benchmarks: decomposing parameters

- CPU Capacity decomposes into:
  - Single CPU performance (depth)
  - SMP scalability (breadth)
  - Measure only one parameter, tune out the others!
- Disk Capacity decomposes into:
  - Per-spindle I/O per second (depth)
  - Channel throughput (breadth)
  - Measure only one parameter, tune out the others!

- Concentrate on CPU capacity for the remainder of this exercise

Decomposing CPU throughput

Throughput is affected by both the speed of the CPU and the number of CPUs.



scaling (assuming no saturation)

Calculate: generate a FOM

- FOM = Figure Of Merit; a measure of relative capacity
- FOM combines benchmarked single CPU performance times an "effective" number of CPUs

$$FOM = \alpha \; Eff(N)$$

α = Relative throughput of a single CPU system

Eff(N) = Effective number of CPUs provided by an N-CPU system

Calculate: α

$$\alpha = \frac{\text{single CPU benchmark for SUT}}{\text{single CPU benchmark for baseline system}}$$

The α factor relates to the single-CPU performance of the SUT (System Under Test). It is a measure of the ratio of the performance of the SUT to the performance of the baseline system

Calculate: Eff(N)

$$\text{Eff(N)} = \frac{\text{SMP benchmark result for N CPUs}}{\text{SMP benchmark result for 1 CPU}}$$

EFF(N), or the Effective CPU factor, relates the performance of a single-CPU configuration of the SUT to an N-CPU configuration of the SUT. It measures how much faster a multi-CPU configuration is in terms of an effective number of CPUs. In almost all cases, Eff(N) < N

Calculate: relating FOM to real capacity

$$\beta = \frac{\text{measured capacity of baseline system}}{\text{FOM of baseline system}}$$

The β factor converts the FOM to a real capacity number (transactions per minute; number of users; megabytes processed…). This is typically the factor that costs the most to determine, since it involves running the full application.

Predict

$$\text{Capacity} = \beta\,(\,\alpha\;\text{Eff}\,(N))$$

Capacity will be in whatever units were used in the baseline benchmarks; transactions per minute, number of simultaneous users, megabytes processed…

Predict: units analysis

$$\text{Capacity} = \beta\,(\alpha\ \text{Eff}\,(N))$$

$$\text{Capacity} = \beta\,(\text{FOM}\,(\text{SUT}))$$

$$\text{Capacity} = \frac{\text{capacity of baseline}}{\text{FOM (baseline)}}\ \text{FOM (SUT)}$$

$$\text{Capacity} = \text{capacity of baseline}\ \frac{\text{FOM (SUT)}}{\text{FOM (baseline)}}$$

Verification

- Baseline system is automatically verified
- Verification should be done periodically
  - Major architectural changes, like conversion from bus to crossbar architecture
  - Major software revisions, like Oracle 7 to 9i
  - Platform changes, like IA32/Linux to IA64/HP-UX
- Verification process:
  1. run benchmarks on system
  2. calculate capacity
  3. run application on system
  4. compare predicted capacity to actual capacity
  5. if they match, great; if not then re-baseline

Example from the Real World

- Telecom software provider
- Performance SLAs are a competitive differentiator
  - "Mercedes S-Class" penalty per hour outage
- Originally predicted capacity based on TPC-A & B
  - Dismal failure due to sparse population of public benchmarks
- Performance parameter:
  - CPU cost (in seconds) per transaction
  - Tantamount to transactions per minute
  - Run a million transactions and time it; divide time by one million times the number of CPUs to get single-CPU cost for each transaction

Example from the Real World cont'd.

- CPU performance decomposition:
  - Single-CPU performance: CDF (natural benchmark)
  - Single-CPU performance: SpecINT (natural benchmark)
  - CPU scalability: SDET (synthetic benchmark)
- Baseline systems:
  - HP 9000 I70
  - HP 9000 K460
  - HP 9000 N4000
- Predictions performed for every single HP 9000 model to date.
- Accuracy within 5% (predicted vs. actual CPU cost per transaction) for every verification

Final Words

- Predicting application performance for capacity planning requires diligence
- Maintaining good capacity planning data pays off for:
  - Software developers and ISVs who deploy multiple copies of a single application
  - Enterprises using templates for deployments of duplicate applications, especially in an adaptive infrastructure
- This process might not be worthwhile for onesies-twosies deployments of custom apps
- There are also some hidden benefits:
  - When performance drops, you know it isn’t due to improper sizing
  - Catches unforseen system performance issues such as OS patch “swiss cheese” effect

Final Words cont'd.

- Take advantage of the HP Capacity Planning Center
  - They'll give you up to two weeks (one week is easier to get scheduled though)
  - They have every machine, every OS revision, and lots of mid- and high-end disk
- Use published benchmarks wisely
  - Some, like SpecINT and SpecFP are quite useful and hard to spoof
  - Know when they're broken (e.g. SpecINT and dual-core chips from an unnamed company)
  - If you can use them, then you don't have to run them yourself

i n v e n t