Performance Measurement and Analysis

9A. Introduction to performance and metrics
9B. Load characterization and generation
9C. Performance Measurement
9D. Performance Analysis
9E. Performance Results Reporting

Performance Analysis Goals

- Quantify the system performance
  – for competitive positioning
  – to assess the efficacy of previous work
  – to identify future opportunities for improvement
- Understand the system performance
  – what factors are limiting our current performance
  – what choices make us subject to these limitations
- Predict system performance
  – how would proposed changes affect performance
- We seek WISDOM ... not numbers!

Principles

- The Pareto Principle
  – 80% of cycles are spent in 20% of the code
- "Data trumps opinions"
  – intuition often turns out to be wrong
  – we can't optimize what we don't measure
- "Rust never sleeps"
  – continuous measurement and comparison
  – if we aren't getting faster, we're getting slower
- Performance is mostly about design
  – code optimization is only occasionally useful

Why performance is so hard

- components operate in a complex system
  – many steps/components in every process
  – ongoing competition for all resources
  – difficulty of making clear/simple assertions
  – systems too large to replicate in laboratory
- lack of clear/rigorous requirements
  – performance is highly dependent on specifics
    - what we measure, how we measure it
  – ask the wrong question, get the wrong answer

Design for Performance

- Establish solid performance requirements
  – justified by technology or competition
  – apportion them to major system components
- Anticipate bottlenecks
  – frequent operations (interrupts, copies, updates)
  – limiting resources (network/disk bandwidth)
  – traffic concentration points (resource locks)
- Design to minimize problems
  – eliminate, reduce use, add resources

Design For Performance Measurement

- Performance is often key to success
  – successful systems generally perform well
  – their performance is constantly improving
- External performance is of limited value
  – it can tell us if performance is good or bad
  – it cannot tell us why we are so performing
- Good measurability must be designed in
  – understand the key diagnostic metrics
  – ensure that each is readily measurable

Performance: what to measure

- competitive performance metrics
  - used to compare competing products
    - nominal response time for simple query
    - standard transactions per second
- engineering performance metrics
  - used to spec components
  - used to analyze performance problems
    - time to perform a particular sub-operation
    - channel utilization, idle time, cycles per operation
- be clear on what your goals are

Metric

a standard unit

- metric must be quantifiable
  - time/rate, size/capacity, effectiveness/reliability …

for measurement or evaluation

- metric must be measurable (or computable)

of something.

- an interesting/valuable quality/characteristic
- metric must be well-correlated with that quality

Choosing Your Metrics

- Core question in any performance study
  - finding the right metrics is half the game
- Pick metrics based on:
  - Completeness: do these metrics span “goodness”?
  - Redundancy: each metric provides new info?
  - Variability: how consistent is it likely to be?
  - Feasibility: can I accurately measure this metric?
  - Diagnostic/Predictive value: yields valuable insight

Common Types of System Metrics

- Duration/ response time
  - Mean latency for a benchmark request?
- Processing rate
  - How many web requests handled per second?
- Resource consumption
  - How much disk is currently used?
- Reliability
  - How many messages delivered without error?
  - Mean Time Between Failure

Sources of Variation in Results

- inconsistent test conditions
  - varying platforms, operations, injection rates
  - background activity on test platform
  - start-up, accumulation, cache effects
- flawed measurement choices/techniques
  - measurement artifact, sampling errors
  - measuring indirect/aggregate effects
- non-deterministic factors
  - queuing of processes, network and disk I/O
  - where (on disk) files are allocated

Capturing Variation

- Generally requires repetition of the same experiment
- Ideally, sufficient repetitions to capture all likely outcomes
  - How do you know how many repetitions that is?
  - You don’t
- Design your performance measurements bearing this in mind

An Example

- 11 pings from UCLA to MIT in one night
- Each took a different amount of time (expressed in msec):

149.1 28.1 28.1 28.5 28.6 28.2
28.4 187.8 74.3 46.1 155.8

- How do we understand what this says about how long a packet takes to get from LA to Boston and back?

Statistical Measures of Samples

- tendency
  - mean ... the average of all samples
  - median ... the value of the middle sample
  - mode ... the most commonly occurring value
- dispersion
  - range ... between the highest and lowest samples
  - standard deviation (σ) ... range for 2/3 of samples
  - confidence interval ... Prob(x is within range)
  - coefficient of variance ... standard deviation/mean

Applied to Our Example Ping Data

- Mean: 71.2
- Median: 28.6
- Mode: 28.1

149.1 28.1 28.1 28.5 28.6 28.2
28.4 187.8 74.3 46.1 155.8

- Which of these best expresses the delay we saw?
  - Depends on what you care about

Applied to Our Ping Data Example

- Range: 28.1,188
- Standard deviation: 62.0
- Coefficient of variation: .87

149.1 28.1 28.1 28.5 28.6 28.2
28.4 187.8 74.3 46.1 155.8

Performance Testing: Factors

"Controlled variations, to enable comparison"

- We do experiments to answer questions
  - trials should be probative of those questions
- Usually we are exploring alternatives
  - what we increased the available memory?
  - what if requests were faster or different?
  - what if we used a different file system?
- Choose factors to explore our questions

Performance Testing: Levels

- A range of values/choices for each factor
- Some factors are boolean:
  - with and without synchronous mirroring
- Some factors have numerical ranges:
  - number of web requests applied per second
  - amount of memory devoted to I/O buffers
- Some factors have categorical levels:
  - Btrfs vs. Ext3 vs. XFS

Choosing Factors and Levels

- Your experiment should look at all key factors
  - each factor tested at each interesting level
- #tests = $\prod$ levels(factor $_i$)
  - this is a minimum if we want to capture variation
  - full range testing may be impractical
- We must choose factors and levels carefully
  - omit some levels of some factors in some tests
  - cover interesting values, but not all combinations

Operations, rates, mixes

- performance is operation-dependent
  - reads, writes, creates, deletes, lookups ...
  - sequential, random, large, small
- it is also operation mix/order-dependent
  - synergistic (e.g. cache) effects
  - adverse (e.g. resource contention) effects
- what mix of operations should we measure
  - what best approximates expected usage?
  - what will best expose strengths and weaknesses

Simulated Work Loads

- Artificial load generation
  - on-demand generation of a specified load
  - controllable operation rates, parameters, mixes
  - scalable to produce arbitrarily large loads
  - can collect excellent performance data
- Weaknesses
  - random traffic is not a usage scenario
  - wrong parameter choices yield unrealistic loads

Captured Sessions

- Captured operations from real systems
  - represent real usage scenarios
  - can be analyzed and replayed over and over
- Weakness
  - each represents only one usage scenario
  - multiple instances not equivalent to more users
  - danger of optimizing the wrong things
  - limited ability to exercise little-used features
  - they are kept around forever, and become stale

Testing under Live Loads

- Instrumented systems serving clients
  - real combinations of real scenarios
  - measured against realistic background loads
  - enables collection of data on real usage
- Weakness
  - demands good performance and reliability
  - potetially limited testing opportunities
  - load cannot be repeated/scaled on demand

Standard Benchmarks

- Carefully crafted/reviewed simulators
  - heavily reviewed by developers and customers
  - believed to be representative of real usage
  - standardized and widely available
  - well maintained (bugs, currency, improvements)
  - comparison of competing products
  - guide optimizations (of benchmark performance)
- Weakness
  - inertia, used where they are not applicable

Meaningful Measurements

- measure under controlled conditions
  - on a specified platform
  - under a controlled and calibrated load
- measure the right things
  - direct measurements of key characteristics
- ensure quality of results
  - competing measurements we can cross-compare
  - measure/correct for artifacts
  - quantify repeatability/variability of results

Common Performance Problems

- non-scalable solutions
  - cost per operation becomes prohibitive at scale
  - worse-than-linear overheads and algorithms
  - queuing delays associated w/high utilization
- bottlenecks
  - one component that limits system throughput
- accumulated costs
  - layers of calls, data copies, message exchanges
  - redundant or unnecessary work

Dealing w/Performance Problems

- is a lot like finding and fixing a bug
  - formulate a hypothesis
  - gather data to verify your hypothesis
  - be sure you understand underlying problem
  - review proposed solutions
    - for effectiveness
    - for potential side effects
  - make simple changes, one at a time
  - re-measure to confirm effectiveness of each
- only harder

End-to-End Testing

- client-side throughput/latency measurements
  - elapsed time for X operations of type Y
  - instrumented clients to collect detailed timings
- advantages
  - easy tests to run, easy data to analyze
  - results reflect client experienced performance
- disadvantages
  - no information about why it took that long
  - no information about resources consumed

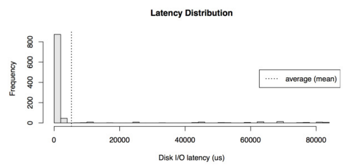
Common Measurement Mistakes

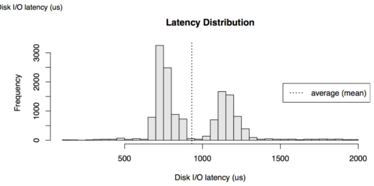
- measuring time but not utilization
  - everything is fast on a lightly loaded system
- capturing averages rather than distributions
  - outliers are usually interesting
- ignoring start-up, accumulation, cache effects
  - not measuring what we thought
- ignoring instrumentation artifact
  - it may greatly distort both times and loads

System Resource Utilization

```
% time io_benchmark_3
real       0m0.178s
user 0m0.003s
sys        0m0.005
% mpstat
07:44:18 CPU %user   %nice   %system  %iowait  %irq   %soft   %idle   intr/s
07:44:18 all  3.01    57.31   0.36     0.13     0.01   0.00    39.19   1063.46
07:44:18 0    5.87    69.47   0.44     0.05     0.01   0.01    24.16   262.11
07:44:18 1    1.79    48.59   0.36     0.23     0.00   0.00    49.02   268.92
07:44:18 2    2.19    42.63   0.28     0.16     0.01   0.00    54.73   260.96
07:44:18 3    2.17    68.56   0.34     0.06     0.03   0.00    28.83   271.47
% iostat -d
Device:    tps      read/s   wrtn/s   read                   wrtn
    sda    194.72   1096.66  1598.70  2719068704             3963827344
    sda1 178.20     773.45   1329.09  1917686794             3295354888
    sda2   16.51    323.19   269.61   801326686  668472456
    sdb    371.31   945.97   1073.33  2345452365             2661206408
    sdb1   371.31   945.95   1073.33  2345396901             2661206408
    sdc    408.03   207.05   972.42   513364213  2411023092
    sdc1   408.03   207.03   972.42   513308749  2411023092
```

Averages Don't Tell the Story





Cache, Accumulation Start-up Effects

- cached results may accelerate some runs
  - random requests that are unlikely to be in cache
  - overwhelm cache w/new data between tests
  - disable or bypass cache entirely
- start-up costs distort total cost of computation
  - do all forks/opens prior to starting actual test
  - long test runs to amortize start-up effects down
  - measure and subtract start-up costs
- system performance may degrade with age
  - reestablish base condition for each test

Measurement Artifact

- costs of instrumentation code
  - additional calls, instructions, cache misses
  - additional memory consumption and paging
- costs of logging results
  - may dwarf the costs of instrumentation
  - increased disk load/latency may slow everything
- make it run-time controllable option
- minimize file/network writes
  - in-memory circular buffer, reduce before writing

Execution Profiling

- automated measurement tools
  - compiler options for routine call counting
    - one counter per routine, incremented on entry
  - statistical execution sampling
    - timer interrupts execution at regular intervals
    - increment a counter in table based on PC value
    - may have configurable time/space granularity
  - tools to extract data and prepare reports
    - number of calls, time per call, percentage of time
- very useful in identifying the bottlenecks

Execution Profiling

Simple execution profiling

%time	seconds	cum %	cum sec	procedure (file)
42.9	0.0029	42.9	0.00	printit (profsample.c)
42.9	0.0029	85.7	0.01	add_vector (profsample.c)
14.3	0.0010	100.0	0.01	mult_by_scalar (profsample.c)

Profiling with call counting

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
42.9	0.0029	0.0029	2200	0.0013	0.0013	printit
42.9	0.0058	0.0029	20	0.1450	0.1450	add_vector
0	0.0058	0.0000	1			main
14.3	0.0068	0.0010	2	0.5000	1.2225	mult_by_scalar

Time Stamped Event Logs

- application instrumentation technique
- create a log buffer and routine
  - call log routine for all interesting events
  - routine stores time and event in a buffer
    - requires a cheap, very high resolution timer
- extract buffer, archive, mine the data
  - time required for particular operations
  - frequency of operations
  - combinations of operations
  - also useful for post-mortem analysis

Time Stamping

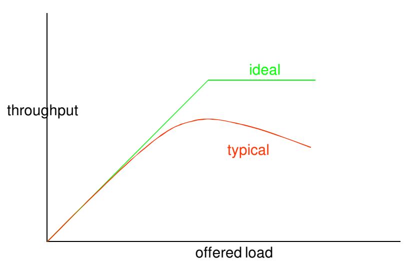
Dump of simple trace log

date	time	event	sub-type
----------	------------	------------	------------
05/11/06	09:02:31.207408	packet_rcv	0x20749329
05/11/06	09:02:31.209301	packet_route	0x20749329
05/11/06	09:02:31.305208	wakeup	0x4D8C2042
05/11/06	09:02:31.401106	read_packet	0x033C2DA0
05/11/06	09:02:31.401223	read_packet	0x033C2DA0
05/11/06	09:02:31.402110	sleep	0x4D8C2042
05/11/06	09:02:31.614209	interrupt	0x00000003
05/11/06	09:02:31.614209	dispatch	0x1B0324C0
05/11/06	09:02:31.614210	intr_return	0x00000003
05/11/06	09:02:31.652303	check_queue	0x2D3F2040
05/11/06	09:02:31.652306	packet_rcv	0x20749329

Performance Analysis

- Can you characterize latency and throughput?
  - of the system, of each major component
- Can you account for all the end-to-end time?
  - processing, transmission, queuing delays
- Can you explain how these vary with load?
- Are there any significant unexplained results?
- Can you predict the performance of a system?
  - as a function of its configuration/parameters

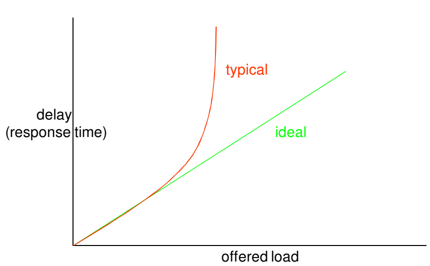
Performance: Throughput vs Load



(why throughput falls off)

- dispatching processes is not free
  - it takes time to dispatch a process (overhead)
  - more dispatches means more overhead (lost time)
  - less time (per second) is available to run processes
- how to minimize the performance gap
  - reduce the overhead per dispatch
  - minimize the number of dispatches (per second)
    - allow longer time slices per task
    - increase the number of servers (e.g. CPUs)
- this phenomenon will be seen in many areas

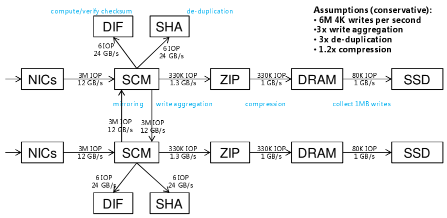
Performance: response time vs load



(why response time grows w/o limit)

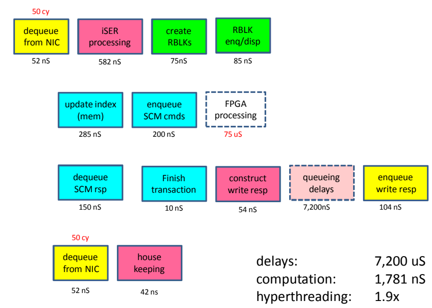
- response time is function of server & load
  - how long it takes to complete one request
  - how long the waiting line is
- length of the line is function of server & load
  - how long it takes to complete one request
  - the average inter-request arrival interval
- if requests arrive faster than they are serviced
  - the length of the waiting list grows
  - and the response time grows with it

Throughput (data flow) model



Assumptions (conservative):
- **6M 4K writes per second**
- **3x write aggregation**
- **3x de-duplication**
- **1.2x compression**

Understanding where the time went



Understanding the Delays

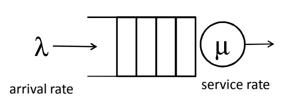
Operation	mean measured queue time	measured CPU % (ρ)	mean measured svc time ($1/\lambda$)	$\lambda\rho^2/(1-\rho)$
4K read	4.1µs	90%	478ns	4.3µs
4K write	2.0µs	88%	267ns	1.9µs

The measured queuing delays within iSER processing very nearly match the values predicted for an M/M/1 system with the measured service times and CPU utilization.

Performance Model Notation

- commonly used concepts/symbols
  - λ request arrival rate (e.g. 200/s)
  - μ request service rate (e.g. 400/s)
  - ρ load factor (λ/μ, e.g. 50%)
- when ($\lambda > \mu$) or ($\rho > 1$)
  - requests arriving faster than they can be serviced
  - the system is overloaded

QT1A: Throughput vs. Latency



$\rho = \lambda/\mu$

utilization/load factor

M/M/1 Queuing System

- Poisson arrivals, FIFO service, one server
- mean queue length: $(1-\rho)/\rho$
- mean waiting time: $\rho/(\mu-\lambda)$

This is a fundamental result

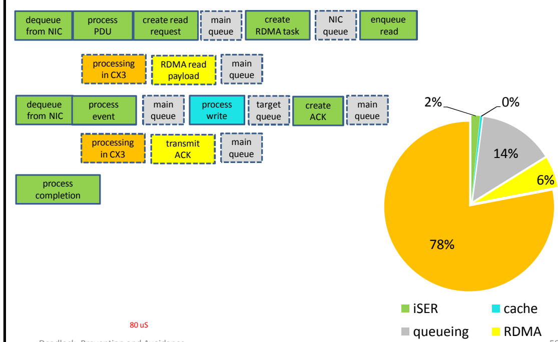
All Presentations

1. To whom am I speaking?
   - what they do, and do not know
   - what they are, and are not prepared to absorb
2. Why are they listening to me?
   - how might this help them achieve their goals
   - how might this address their concerns
3. What do I want them to leave with?
   - what conclusions do I want them to draw
   - what actions do I want them to take

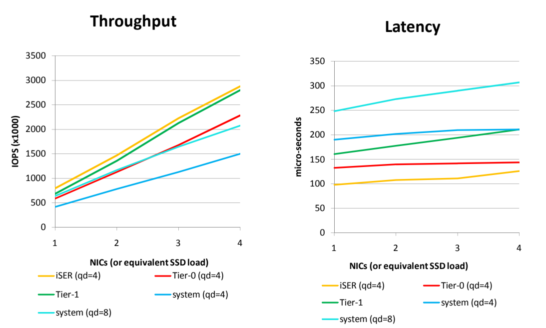
Performance Presentation

- highlight the key results
  - answers to the basic questions
  - identified problems, risks and opportunities
- why should they believe these results
  - methodology employed, relation to other results
  - back-up details
- not just numbers, but explanations
  - how do we now better understand the system
  - how does this affect our plans and intentions

Time Breakdown (high level)



Throughput and Scalability



Sample Conclusions

- **Throughput**
  - iSER throughput linear with NICs (up to limits we could test)
  - cache throughput limited by memory speed (due to large index)
- **Latency**
  - dominated by NIC and queuing delays (not processing time)
    queuing delays are a result of high CPU utilization
    NIC associated delays may be a load-related problem in CX-3
  - very good until increasing queue depth becomes the problem
- **Efficiency and Hyper-Threading**
  - 2-2.5µs of processing per 4K write/read operation
  - NIC/protocol handling hyper-threads very well (1.8x)
  - cache hyper-threading (1.2x-1.4x) is limited by large index

Assignments

- Reading
  - AD Ch 33 events
  - AD Ch 35 introduction to storage
  - AD Ch 36 devices
  - AD Ch 37 disks
  - AD ch 38 RAID
  - poll(2), select(2), sigaction(2)
- Lab
  - Project 4B: embedded system sensor I/O

Supplemental Slides

Performance Testing

- identify key performance metrics
  - throughputs, response times, failure rates
  - some may be external competitive numbers
  - some may be internal assessment numbers
- define ways to measure each
  - test transactions and measurement points
- define suites to exercise and measure
  - there are often performance benchmarks
- this testing should be automated

Meaningful Measurements

- measure under controlled conditions
  - on a specified platform
  - under a controlled and calibrated load
- measure the right things
  - direct measurements of key characteristics
- ensure quality of results
  - competing measurements we can cross-compare
  - measure/correct for artifacts
  - quantify repeatability/variability of results

Factors in Experiments

- We do experiments to answer questions
  - choose factors that are probative of those questions
- If you care about web server scaling …
  - , factors probably related to amount of work offered
- If you want to know which file system works best for you, factor is likely to be different file systems
- If you're deciding how to partition a disk, factor is likely to be different partitionings

Measurement Workloads

- Most measurement programs require the use of a *workload*
- Some kind of work applied to the system you are testing
  - Preferably similar to the work you care about
- Can be of several different forms
  - Simulated workloads
  - Replayed trace
  - Live workload
  - Standard benchmarks