

Performance

- 9A Performance Principles and Metrics
- 9B Load Characterization & Generation
- 9C Performance Measurement
- 9D Performance Analysis

Performance Analysis Goals

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

Deadlock, Prevention and Avoidance

2

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”
 - if we aren’t getting faster, we are getting slower
 - continuous measurement and comparison
- Performance is mostly about design
 - recognizing and mitigating costs and bottlenecks
 - code optimization is only occasionally useful

Deadlock, Prevention and Avoidance

3

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

Deadlock, Prevention and Avoidance

4

Design for Performance

- Establish solid performance requirements
 - justified by technology or competition
 - apportion them to major system components
- Anticipate costs and 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

System Testing and Performance

5

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

System Testing and Performance

7

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

Deadlock, Prevention and Avoidance

8

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?
 - measured from/on client or server side?
- 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 (or Failures in Time)

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

Deadlock, Prevention and Avoidance

11

Capturing Variation

- Generally requires repeating same experiment
- Ideally, enough to capture all likely outcomes
 - How do you know how many repetitions that is?
 - You don't ... until you see the distribution
- Design your tests bearing this in mind
 - enough to capture the range of variation
 - but manage the combinatoric explosion

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 dev/mean

Deadlock, Prevention and Avoidance

14

Applied to Our Example Ping Data

- Mean: 71.2
- Median: 28.6 149.1 28.1 28.1 28.5 28.6 28.2
28.4 187.8 74.3 46.1 155.8
- Mode: 28.1
- Which of these best expresses our delays
 - Depends on what you care about

Applied to Our Ping Data Example

- Range: 28.1,188
- Standard deviation: 62.0
- Coefficient of variation: .87 ($\sigma \approx \text{mean}$)

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
 - some factors are likely to interact with others
- #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

Deadlock, Prevention and Avoidance

20

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

Deadlock, Prevention and Avoidance

21

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

Deadlock, Prevention and Avoidance

22

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
 - potentially limited testing opportunities
 - load cannot be repeated/scaled on demand

Deadlock, Prevention and Avoidance

23

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

Deadlock, Prevention and Avoidance

24

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 ... because they are not failures

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
 - merely a score, unlikely to yield understanding

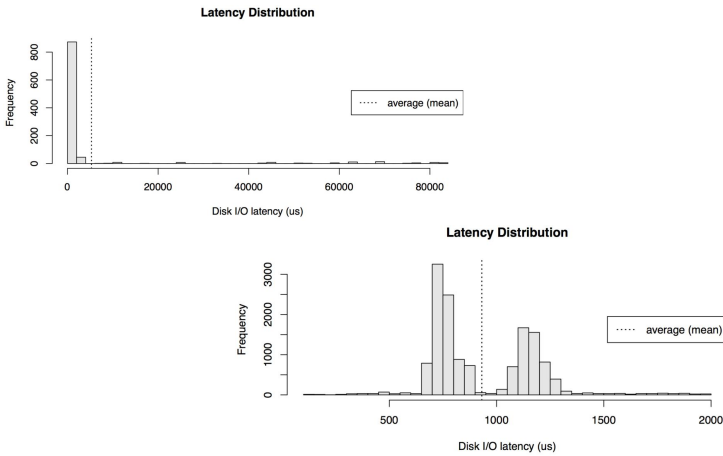
Common Measurement Mistakes

- 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 0m0.005s
% 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

% cumulative	self	self	total			
time	seconds	seconds	calls	ms/call	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

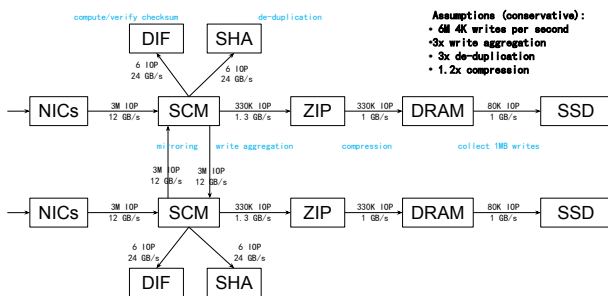
Dump of simple trace log

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

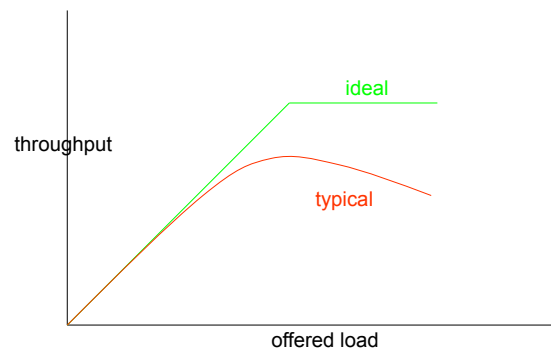
Performance Analysis

- 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

Throughput (data flow) model



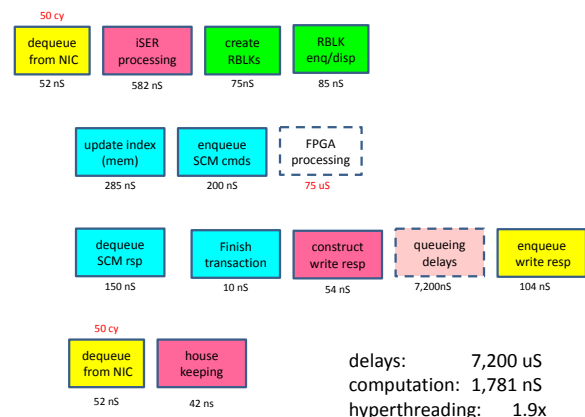
Performance: Throughput vs Load



(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

Understanding where the time went

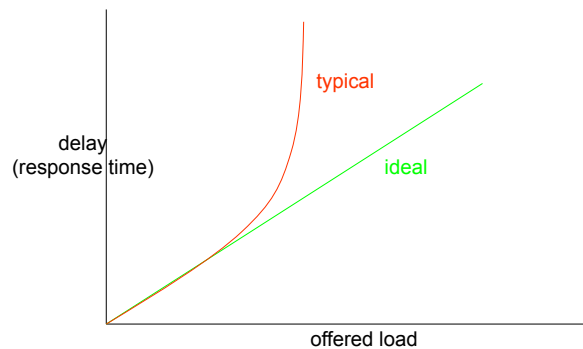


Understanding the Delays

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

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

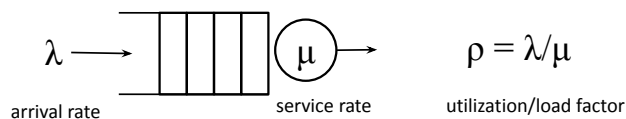
Performance: response time vs load



(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

QT1A: Throughput vs. Latency



M/M/1 Queuing System

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

This is a fundamental result

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

Reading and Assignments

Reading:

- Arpaci C27.4 ... condition variables
- Arpaci C28.12-15 ... spin waits
- Arpaci C30-30.1 ... condition variables

Projects:

- we will help w/project 2B problems in the lab