Performance Measurement and Analysis

9A. Introduction to performance and metrics
9B. Load characteristics and generation
9C. Performance Measurement
9D. Performance Analysis
9E. Performance Presentation

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!

Performance Measurement and Analysis 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 measurement and analysis

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

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 ($\sigma$) ... range for 2/3 of samples
  - confidence interval ... $\text{Prob}(x \text{ is within range})$
  - coefficient of variance ... standard deviation/mean

Applied to Our Example Ping Data

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

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 \text{levels(factor)}
  - 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
  - potentially 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

```plaintext
<table>
<thead>
<tr>
<th>Device</th>
<th>iops</th>
<th>read/4K</th>
<th>write/4K</th>
<th>read time</th>
<th>write time</th>
</tr>
</thead>
<tbody>
<tr>
<td>sda</td>
<td>194.72</td>
<td>1084.66</td>
<td>1188.70</td>
<td>2729358794</td>
<td>3963827344</td>
</tr>
<tr>
<td>sda1</td>
<td>178.20</td>
<td>773.45</td>
<td>1329.09</td>
<td>1917686794</td>
<td>3295354888</td>
</tr>
<tr>
<td>sda2</td>
<td>16.51</td>
<td>323.19</td>
<td>269.61</td>
<td>801326686</td>
<td>668472456</td>
</tr>
<tr>
<td>sdb</td>
<td>371.31</td>
<td>945.97</td>
<td>1073.33</td>
<td>2345452365</td>
<td>2661206408</td>
</tr>
<tr>
<td>sdb1</td>
<td>371.31</td>
<td>945.95</td>
<td>1073.33</td>
<td>2345396901</td>
<td>2661206408</td>
</tr>
<tr>
<td>sdc</td>
<td>408.03</td>
<td>207.05</td>
<td>972.42</td>
<td>5133641213</td>
<td>2411023092</td>
</tr>
<tr>
<td>sdc1</td>
<td>408.03</td>
<td>207.03</td>
<td>972.42</td>
<td>5133641213</td>
<td>2411023092</td>
</tr>
</tbody>
</table>
```

Performance measurement and analysis
Averages Don’t Tell the Story

- 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

Cache, Accumulation Start-up Effects

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

Simple execution profiling

<table>
<thead>
<tr>
<th>Procedure (File)</th>
<th>% Time</th>
<th>Cumulative %</th>
<th>Self Time</th>
<th>Cumulative Self Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>printit (profample.c)</td>
<td>42.9</td>
<td>42.9</td>
<td>0.0029</td>
<td>0.0029</td>
</tr>
<tr>
<td>add_vector (profample.c)</td>
<td>14.3</td>
<td>57.2</td>
<td>0.0010</td>
<td>0.0039</td>
</tr>
</tbody>
</table>

Profiling with call counting

<table>
<thead>
<tr>
<th>Time</th>
<th>Seconds</th>
<th>Calls</th>
<th>Total Time</th>
<th>Self Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>42.9</td>
<td>0.0029</td>
<td>2000</td>
<td>0.0013</td>
</tr>
<tr>
<td>100</td>
<td>14.3</td>
<td>0.0010</td>
<td>1000</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

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
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

Dump of simple trace log

<table>
<thead>
<tr>
<th>date</th>
<th>time</th>
<th>event</th>
<th>sub-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>05/11/06</td>
<td>09:02:31.207408</td>
<td>packet_recv</td>
<td>0x20749329</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.209301</td>
<td>packet_route</td>
<td>0x20749329</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.305208</td>
<td>wakeup</td>
<td>0x408C2042</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.401106</td>
<td>read_packet</td>
<td>0x633C2DA0</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.401223</td>
<td>packet_route</td>
<td>0x333C2DA0</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.402110</td>
<td>sleep</td>
<td>0x408C2042</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.614209</td>
<td>interrupt</td>
<td>0x00000003</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.614209</td>
<td>dispatch</td>
<td>0x180324C0</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.614210</td>
<td>intr_return</td>
<td>0x60400003</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.652303</td>
<td>check_queue</td>
<td>0x203F2040</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.652306</td>
<td>packet_recv</td>
<td>0x20749329</td>
</tr>
</tbody>
</table>

Performance: Throughput vs Load

- 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 throughput 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

- 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

Understanding where the time went

Understanding the Delays

Performance Model Notation

QT1A: Throughput vs. Latency

All Presentations
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 (may not plan on presenting)
• not just numbers, but explanations
  – how do we now better understand the system
  – how does this affect our plans and intentions

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

• Projects
  – finish Project 1B (contention)
• Reading (55pp)
  – AD 33-33.6 (events)
  – AD 35 (introduction to storage)
  – AD 36 (devices)
  – AD 37 (disks)
  – AD 38 (RAID)
  – Device Drivers, Classes, and Services
  – Dynamically Loadable Drivers

Charles Dickens on System Performance

“Annual income, twenty pounds;
annual expenditure, nineteen, nineteen, six;
Result ... happiness.
Annual income, twenty pounds;
annual expenditure, twenty pounds ought & six;
Result ... misery!”

Wilkins Micawber, David Copperfield