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

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

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

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)
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

Performance Testing

- identify key performance metrics
  - throughputs, response times, capacities
  - 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

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

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

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

Common Performance Problems

- non-scalable solutions
  - cost per operation becomes prohibitive at scale
  - worse-than-linear overheads and algorithms
  - queuing delays associated with 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

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:

<table>
<thead>
<tr>
<th>%time</th>
<th>seconds</th>
<th>%</th>
<th>cum sec</th>
<th>procedure (file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.9</td>
<td>0.0029</td>
<td>42.9</td>
<td>0.00</td>
<td>printit (profsample.c)</td>
</tr>
<tr>
<td>42.9</td>
<td>0.0029</td>
<td>85.7</td>
<td>0.01</td>
<td>add_vector (profsample.c)</td>
</tr>
<tr>
<td>14.3</td>
<td>0.0010</td>
<td>100.0</td>
<td>0.01</td>
<td>mult_by_scalar (profsample.c)</td>
</tr>
</tbody>
</table>

Profiling with call counting:

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>self seconds</th>
<th>total seconds</th>
<th>calls</th>
<th>ms/call</th>
<th>ms/call name</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.9</td>
<td>0.0029</td>
<td>0.0029</td>
<td>2200</td>
<td>0.0013</td>
<td>printit</td>
</tr>
<tr>
<td>42.9</td>
<td>0.0058</td>
<td>0.0029</td>
<td>20</td>
<td>0.1450</td>
<td>add_vector</td>
</tr>
<tr>
<td>0</td>
<td>0.0058</td>
<td>0.0000</td>
<td>1</td>
<td></td>
<td>dispatch</td>
</tr>
<tr>
<td>14.3</td>
<td>0.0068</td>
<td>0.0010</td>
<td>2</td>
<td>0.5000</td>
<td>mult_by_scalar</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

Time Stamping

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_rcv</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>09:02:31.401223</td>
<td>read_packet</td>
<td>0x033C20A0</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.401110</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>0x033C20A0</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.614209</td>
<td>intr_return</td>
<td>0x00000003</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31.652303</td>
<td>check_queue</td>
<td>0x20749329</td>
</tr>
</tbody>
</table>

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:

Understanding where the time went:

- Assumptions (conservative):
  - 6M 4K writes per second
  - 3x write aggregation
  - 3x de-duplication
  - 1.2x compression
- Compute/verify checksum de-duplication
- Write aggregation mirroring compression
- Collect 1MB writes

- Delays: 7,200μs
- Computation: 1,781 nS
- Hyperthreading: 1.9x
- Enqueue write resp: 54 nS
- Transaction completion: 10 nS
- Enqueue SCM cmd: 200 nS
- Enqueue RBLK: 75 nS
- Update index (mem): 582 nS
- Computation: 1,781 nS
Understanding the Delays

<table>
<thead>
<tr>
<th>Operation</th>
<th>mean measured queue time</th>
<th>measured CPU % (ρ)</th>
<th>mean measured svc time (1/λ)</th>
<th>λρ²/(1-ρ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K read</td>
<td>4.1µs</td>
<td>90%</td>
<td>478ns</td>
<td>4.3µs</td>
</tr>
<tr>
<td>4K write</td>
<td>2.9µs</td>
<td>88%</td>
<td>267ns</td>
<td>1.9µs</td>
</tr>
</tbody>
</table>

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.

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

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
  - 2.2-2.5µs 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

Throughput and Scalability

Time Breakdown (high level)
Principles

• The Pareto Principle
  – 80% of cycles are spent in 20% of the code
• We need real data
  – we can’t optimize what we don’t measure
  – intuition often turns out to be wrong
• Performance demands eternal vigilance
  – 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

Design for Performance

• Establish performance requirements
• 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
• Include performance measurement in design
  – what will be measured, and how

Assignments

• for the next lecture:
  – Arpaci ch 33-33.6 Asynchronous I/O
  – Arpaci ch 36 I/O Devices
  – Arpaci ch 37 Hard Disk Drives
  – Arpaci ch 38 Redundant Disk Arrays (RAID)
  – Device Drivers, Classes and Services
  – Dynamically Loadable Drivers