

System Testing & Performance

- System Testing
 - system testing vs. unit testing
 - planning for system testing
- Bug Finding
 - bug detection rates
 - release phases & ship criteria
- Performance Management
 - basic principles
 - performance design and repair
 - basic tools and practices

System Testing and Performance

1

Unit vs. System Testing

- Goals
 - is this component ready to integrate?
 - is this product ready to ship?
- Context
 - testing components in (relative) isolation
 - testing the entire assemblage
- Focus
 - component functionality and specifications
 - system functionality and specifications
 - “whole system” behavior

System Testing and Performance

2

Testing Responsibilities

- There are multiple players
 - developers
 - testers working with the developers
 - independent Test Group
- There are many types of testing
 - component functional validation
 - system functional validation
 - usability, security, performance, robustness
- Many theories about who should do which
 - organizational, philosophical, process, moral

System Testing and Performance

3

Typical System Test Activities

- System functionality and error handling
 - does the system do the things it is supposed to do
 - including the correct handling of all specified error conditions
- Installation (and upgrade) testing
 - do all of the parts install and configure correctly on all platforms
 - do upgrades properly preserve all persistent state
- Usability testing
 - can ordinary people figure out how to use it
- Security testing
 - authentication, authorization, privacy, attacks
- Interoperability testing
 - platforms, devices, different clients & servers
- Performance and scalability testing
 - whole system capacity, throughput, response time
- Stress testing ... ideally in random combinations
 - whole system overload, resource exhaustion, error recovery

System Testing and Performance

4

Planning System Testing

- Some system testing is fairly obvious
 - does the whole system meet specified requirements
 - derive test cases from the requirements
 - fully exercise independent component interactions
 - derive test cases from common use-case scenarios
 - derive test cases from architecture and specifications
- But what other tests do we need to include?
 - standard types of testing:
 - installation, usability, security, interoperability, performance
 - other types of product or domain specific testing?
- System Testing is about confidence
 - what will the product be expected to do?
 - what are we already confident that it can do?
 - what are we still not yet sure of?

System Testing and Performance

5

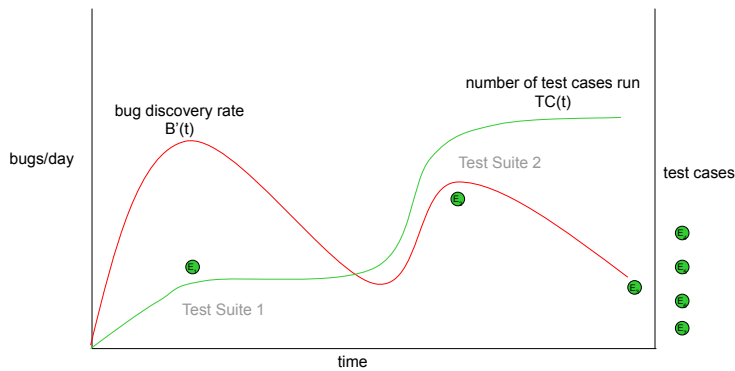
Bug Discovery Rate

$$M(t) = A * C(t) * K$$

- $M(t)$... number of defects found by time t
- A ... number of actual defects in code
- $C(t)$... total space coverage by time t
- K ... efficacy of this testing technique

6

Testing and Bug Discovery



System Testing and Performance

7

Basic Principles of Performance

- The Pareto Principle
 - 80% of cycles are spent in 20% of the code
 - (my experience is more like 90% and 2%)
- Performance requires real measurement
 - our intuition usually turns out to be wrong
 - cannot optimize what we do not measure
- Performance demands eternal vigilance
 - if we aren't getting faster, we're getting slower
- Performance is mostly about design
 - code optimization is only occasionally useful

System Testing and Performance

8

Design for Performance

- Establish performance requirements
- Anticipate bottlenecks
 - frequent operations (copies, updates, messages)
 - limiting resources (network/disk bandwidth)
 - traffic concentration points (resource locks)
- Design to minimize problems
 - eliminate, reduce use, add resources
 - and, sometimes, optimize implementation
- Include performance in design reviews

System Testing and Performance

9

Fixing 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

System Testing and Performance

10

Team Exercise

1. consider your component unit test cases
 - any un-covered *whole product* functionality?
 - can you define additional system testing?
 - other (e.g. install, config, security) issues?
2. consider computation, I/O, services
 - might any involve significant cost/delay?
 - how could you measure those costs?
 - new thoughts on design or testing?
3. be prepared to discuss your results

11

For Next Lecture

- McConnell, section 34.9
 - eschew religious wars
- Wikipedia: Agile Software Development
- Wikipedia: SCRUM
 - Roles
 - Sprint
 - Meetings
 - Artifacts
- Github Project Boards

Team Structure and Processes

12

Supplementary Slides

Pre-FCS Exposure

- give key customers an early sniff
 - so they can better prepare for new release
 - so they can start work on related products
- get additional feedback prior to FCS
 - real customers will use it in different ways
 - they will find different bugs
 - they will find usability issues
 - they will find compatibility issues
- obscure the fact we are behind schedule
 - we shipped something!

Alpha Testing

- alpha products are usually incomplete
 - missing functionality and documentation
 - very buggy
 - non-standard installation and management
- alpha sites are carefully selected
 - they are prepared to deal with problems
 - they can be trusted to exercise the product
- goals of alpha testing
 - gather feedback on key features and content
 - early access for partners, key customers

Beta Testing

- a beta product is very near final form
 - all functionality and documentation complete
 - few significant known problems
- beta sites are real customers
 - they will put the product to real use
 - they have agreed to give us feedback
- goals of beta testing
 - confirm the product is ready to ship
 - gather last-minute feedback
 - early access for key customers

determining ship-ability

- many types of metrics are commonly used
 - list of tests that must be passed
 - hours of load/stress tests passed
 - performance criteria
 - beta report results
 - open bug counts
 - new bug arrival rates
- concurrence (engineering, Q/A, support)
- fiat (product executive says “ship it”)
- criteria must be set at start of project

In-System Unit Testing

- test new component in a whole system
 - other components with which to interact
 - we are able to exercise more functionality
 - we can identify component interface issues
- add new component to existing system
 - where other components are known good
- test builds of experimental components
 - tracking down problems can be difficult
- this is still unit, and not system testing

Functionality Testing

- does system function as specified?
 - test all specified functional requirements
 - these are part of the acceptance criteria
- may involve a large number of test cases
 - many may be existing unit test cases
 - may include standards conformance suites
 - may include whole-system exercises
 - may include sample scenarios
- this testing should be automated

Robustness Testing

- introduce each specified error
 - invalid options, requests, and data
 - communications errors, resource failures, ...
- verify correct system response
 - check that system properly detects it
 - check that system properly reports it
 - check that system properly responds to it
 - check that system continues working
- this testing should be automated

Installation Testing

- installation is functionality too
 - component install may be checked w/unit test
- whole system install also needs testing
 - follow directions and use the defaults
 - try all of the options and combinations
 - is feedback correct
 - is correct software installed correctly
 - does upgrade work (preserving old data)
- some automated testing, some manual

Usability Testing

- most testing should be automated
 - so that it can be run frequently
 - so that it is run consistently
- this should be complemented with usage
 - users sit down and try to use the software
 - they try do do normal things
 - they try to make obvious mistakes
 - such testing turns up many problems
 - but at some point, the pay-off falls off

Security Testing

- much of this is specified functionality
 - authentication/authorization mechanisms
 - data protection mechanisms
 - response to unauthorized requests
 - covered by automated functionality test cases
- it is also necessary to look for holes
 - design review by domain security experts
 - penetration tests by experienced hackers

Interoperability Testing

- consider everything system depends on
 - hardware, operating system, other services
 - there are probably multiple versions of each
- consider everything system talks to
 - browsers, servers, routers, etc.
 - there are probably multiple versions of each
- we should test on all combinations
 - or one from each equivalence partition
 - defining these requires insight an experience

Stress Testing

- bugs are often found in special cases
 - resource exhaustion, error handling
 - unlikely combinations of events
- stress tests create these continuously
 - traffic generators running at full capacity
 - with changing random mixes of requests
 - continuous error generation
 - with random error selection
- stress tests can run for days or longer
 - they shake out many hard-to-cause problems

Load Generation

- system specifications include capacity
 - support up to 3000 transactions/second
- performance should be measured at load
 - response time a 1500 transactions/second
- many bugs involve concurrent operations
 - locking, allocate/free, protection, etc.
- these require automatic load generation
 - generate traffic of specified types
 - generate traffic at calibrated rates

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
- write suites to exercise and measure
 - there are often performance benchmarks
- this testing should be automated

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
- must be meaningful and well defined

Perf: meaningful measurements

- measure under controlled conditions
 - on a specified platform
 - under a controlled and calibrated load
 - without other conflicting activities
- ensure validity of results
 - measuring very brief operations
 - ultra-high resolution timers
 - perform a large number of operations
 - repeatability of results
 - collect many samples, explain variations

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 | self 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_vecto |
| 14.3 | 0.0068 | 0.0010 | 2 | 0.5000 | 1.2225 | mult_by_scalar |
| 0 | 0.0058 | 0.0000 | 1 | | | main |

Time Stamping

- 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

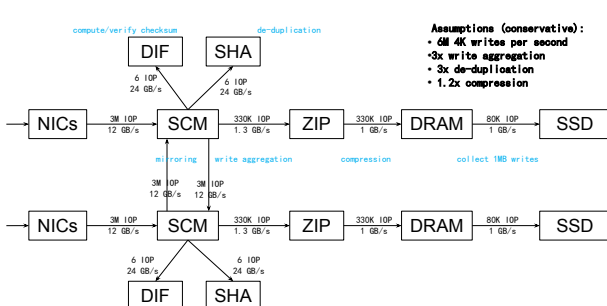
Time Stamping

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 |

- 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 the Delays

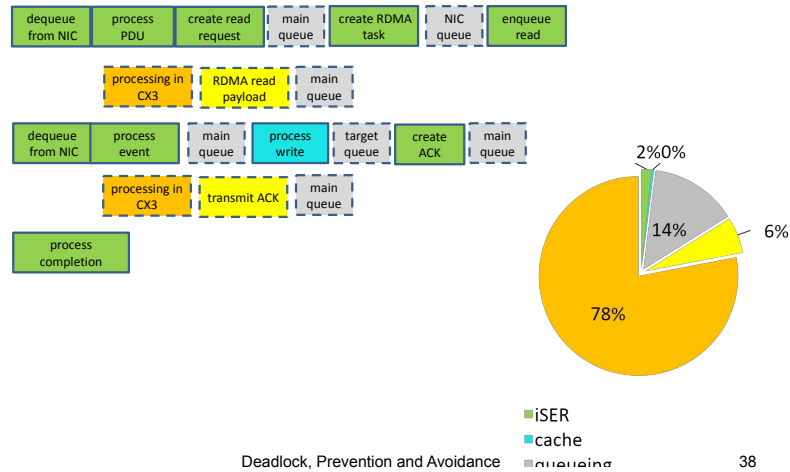
| Operation | mean meas queue time | meas CPU % (ρ) | mean meas svc time ($1/\lambda$) | $\lambda\rho^2/(1-\rho)$ |
|-----------|----------------------|-----------------------|------------------------------------|--------------------------|
| 4K read | 4.1 μ s | 90% | 478ns | 4.3 μ s |
| 4K write | 2.0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

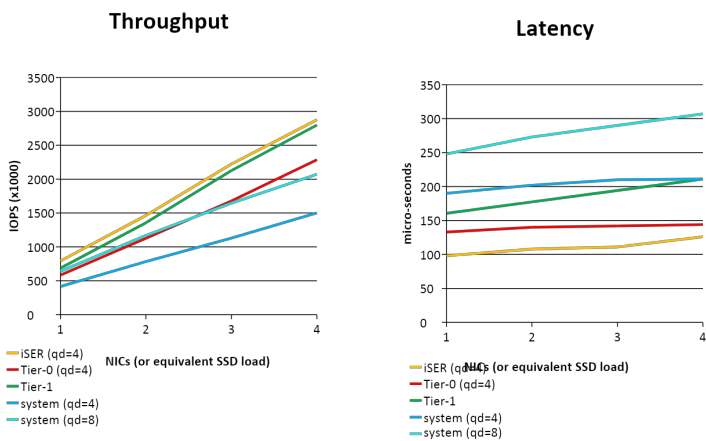
Performance Presentation

- 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