Mitigating Performance Risk

Four Steps to Releasing with Confidence

RBCS

TIME TESTED.
TESTING IMPROVED.

www.RBCS-US.com

Introduction

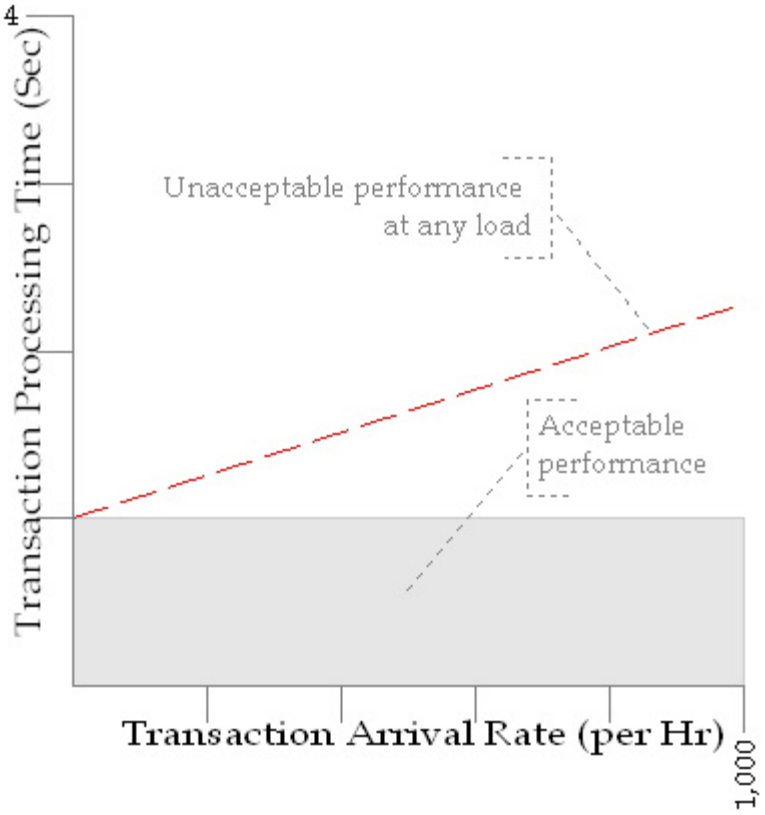
- Have you worked on a project where performance was the last item on the agenda, and it blew up in everyone's face?
- Ever seen projects cancelled due to performance problems?
- Nasty end-of-project performance surprises are avoidable
- It's not complicated, but it is hard work
- Let's review some typical performance problems, then a four step process you can use to avoid performance disasters

Typical Performance Problems

- Often, one or more of the following performance problems are observed during performance testing
  - Slow response under all load levels
  - Slow response under moderate-to-heavy (but supposedly allowed) load levels
  - Degraded response over time
  - Inadequate or graceless error handling under heavy or over-limit load
- Let's take a look…

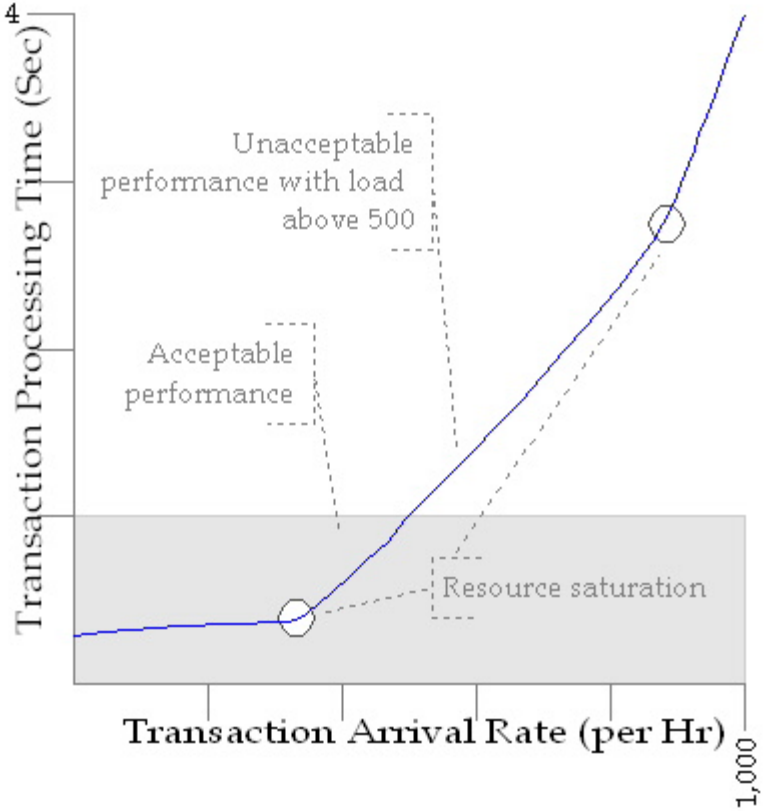
Slow Response Under All Load Levels

- In some cases, response is unacceptable regardless of load
- Underlying bugs include:
  - Bad database design or implementation
  - Network latency
  - Other background loads
- These problems can be detected under regular functional testing, not just performance testing



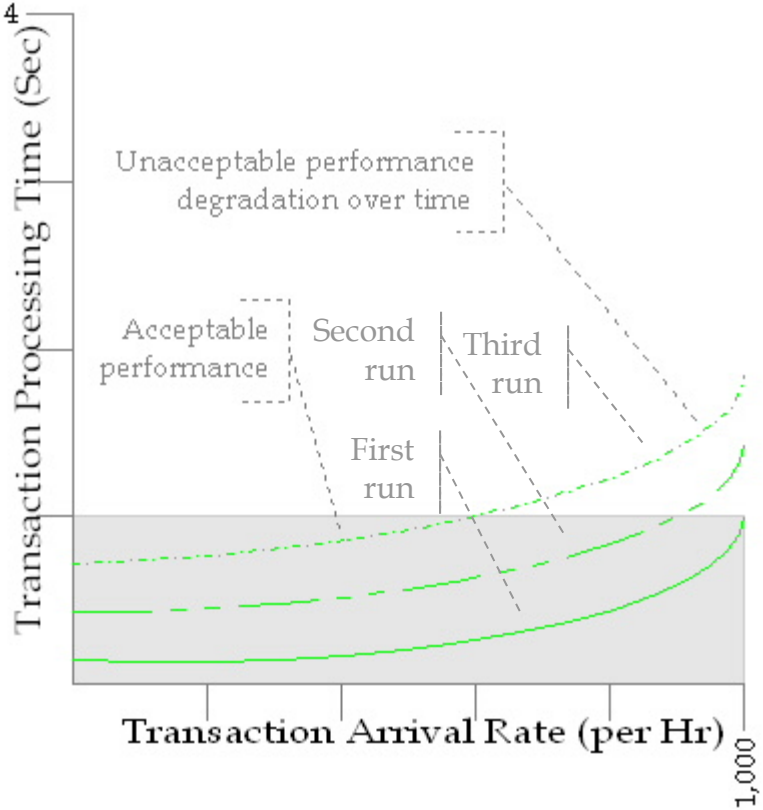
Slow Response At Moderate-to-Heavy Load

- In some cases, response degrades unacceptably with moderate-to-heavy load
- Underlying bugs include:
  - Saturation of one or more resources
  - Varying background loads



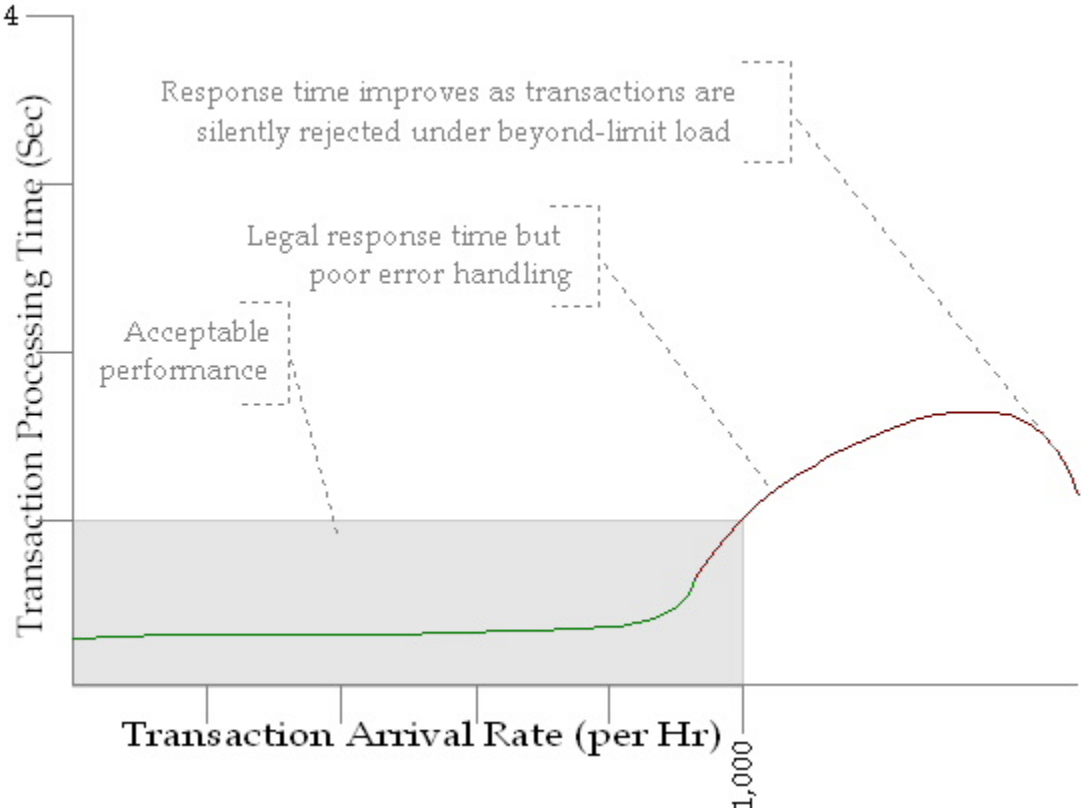
Degraded Response Over Time

- In some cases, response degrades over time
- Underlying bugs include:
  - Memory leaks
  - Disk fragmentation
  - Increasing network load over time
  - Database growth



Graceless Error Handling Under Load

- In some cases, response time is acceptable, but error handling degrades at high and beyond-limit load
- Underlying bugs include:
  - Insufficient resource pools
  - Undersized queues and stacks
  - Too-rapid time-out settings



The Case Study

- This case study involves testing a set of cloud servers supporting Internet appliances
- A key quality risk was that these servers might not handle anticipated loads
- The project was planned to last about six months
- The four steps to manage performance risk:
  1. Static performance tests of server designs
  2. Performance simulation of server designs
  3. Unit performance tests during implementation
  4. System performance tests of the implemented system
- Most organizations skip steps 1-3 and begin step 4 just before installation

Static Performance Testing

- At the start, the server system architects built a model of system behavior
- Model built in a spreadsheet
  - Estimated resource utilization levels based on number of deployed Internet appliances
  - Estimated resource utilization included CPU, memory, network bandwidth, etc.
- Model was peer reviewed by testers, developers, and managers
- The model was adjusted until we were confident in the initial design

Performance Simulation

- Next, the server system architects created a dynamic simulation
- This required diverse skills in network and internetwork architecture, programming, system administration, and the use of the simulation tool
- The simulation was fine-tuned iteratively until the architects had confidence in its predictions
- Purchasing hardware and configuring the servers also were iterative
- Once confident in the model's prediction of the need for a certain element, we would purchase it

Unit Performance Testing

- The system consisted of ten servers providing five key capabilities:
  - E-commerce hosting
  - E-mail repository
  - IMAP mail
  - Various user-related databases
  - Software and operating system updates.
- Load balancing split each key capability's work across two servers
- Each of these five capabilities underwent unit testing, with a unit defined as a distinct service provided as part of the capability
- Developers used some internally developed tools to which we added some enhancements

System Performance Testing

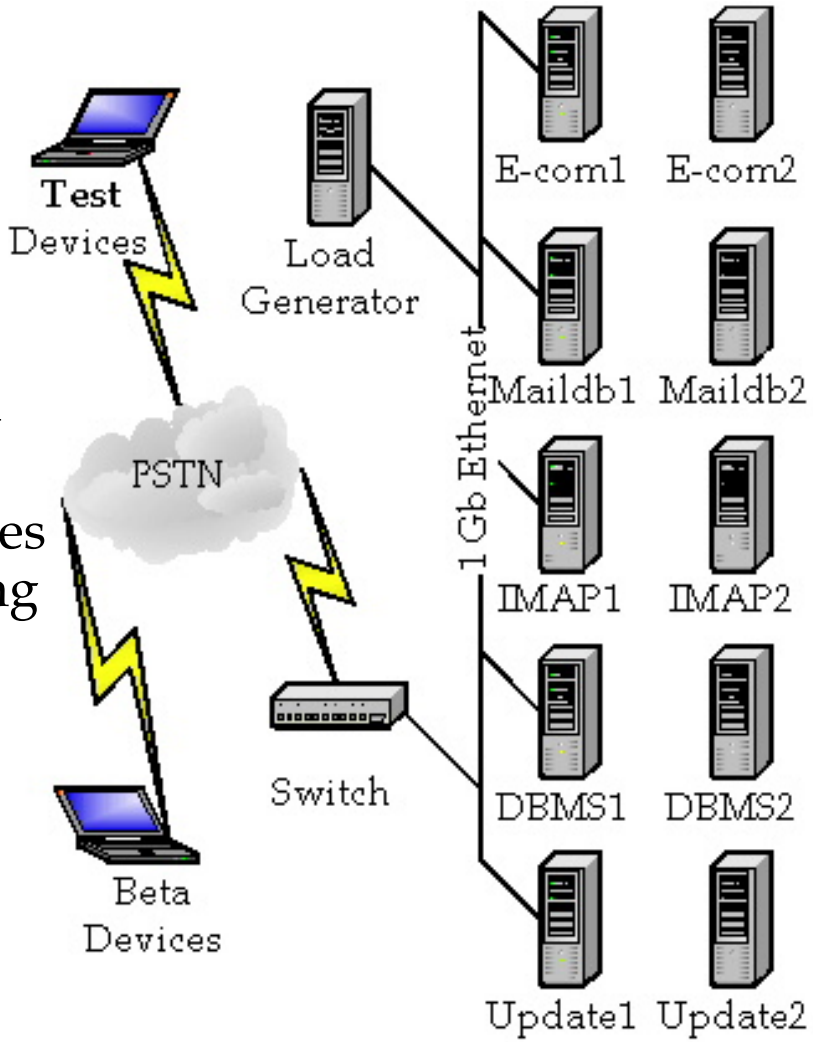
- During development and unit performance testing, we planned and prepared for system testing, including performance testing
- Performance testing had its own test plan, with the following main objectives:
  - To meet and exceed real-world, realistic loads
  - To test our load test tools
  - To provide early, quick feedback about performance bugs
  - To validate the performance simulation
  - To test robustness and recovery by forcing system and component failures
- We also planned to iterate tests expeditiously and efficiently as bugs were found and fixed

Features Tested

- Update: The appliances were updated by update servers
  - We built a load generator from unit test tools to simulate thousands of appliances
  - The simulated appliances requested updates, loading update servers
- Mail: The clients sent and received e-mail via IMAP e-mail servers
  - We simulated client e-mail traffic, including attachments
  - Emails varied in size
- Web: When an appliance was sent to a customer, the servers would set up and support its features via web services, so we simulated realistic loads of activity
- Database: Information about each appliance was stored in database servers, so we simulated various database activities, often using SQL scripts
- Our usage scenarios, assumptions, and plan were reviewed with developers throughout the test preparation process

Performance Test Environment

- We built and validated the test environment during preparation
- Load balancing devices are not shown but are part of the configuration
- We were able to use the production servers as the test environment
- We had both client testing appliances and beta testing appliances accessing the servers
- Sometimes, we ran other functional and non-functional tests during performance testing to get a subjective impression of the users' experience



So, What Did We Find?

- Once system testing started, we ran tight cycles of performance testing, getting fixes regularly
- We found some interesting server performance bugs:
  - Sending corrupted update files hung the update process
  - Large log files caused performance problems
  - Network management agents failed silently when server crashes were simulated
  - Memory leaks degraded performance
  - The database was configured improperly
  - Hardware was not hot-swappable
  - Throughput was too low on critical processes
  - Load balancing did not work properly
  - Proxy and filtering bugs caused slow Internet access
- Most of these bugs couldn't have been found in steps 1-3
- All were manageable code problems, not fundamental design issues
- Let's look examples of some e-mail bugs we found…

At First Glance…

- Simulation results for the IMAP e-mail servers
  - 45% server utilization at 25,000 users
  - 75% server utilization (saturation) at 40,000 users
- Worst-case snapshots (25,000 users)

Server	CPU Idle
MTA1	68%
MTA2	79%
Maildb1	67%
Maildb2	89%
IMAP1	59%
IMAP2	55%

∴ Test results appear to support simulation results at 25,000 users, but…

…But Actually…

We did a detailed analysis of performance, too

Transaction	Server	Frequency	Simulation Time (ms)	Observed Time (ms)
Connect	IMAP	1 per connect	0.74	0.77
Banner	IMAP	1 per connect	2.57	13.19
Authorize	IMAP	1 per connect	19.68	19.08
Select	IMAP	1 per connect	139.87	163.91
Fetch	IMAP	1 per connect	125.44	5,885.04

To do this, you must design your tests and models so that you can compare the results

Conclusions

- Emergent system behaviors like performance can blow up your project at the very end…if you let them
- However, hard, upfront work on these four basic steps can reduce the risk to a manageable level
- Resource analysis, simulation, unit testing, and system testing each filter defects, including performance defects
- Don't gamble with your project's future by waiting until the end to address performance

…Contact RBCS

For over two decades, RBCS has delivered consulting, outsourcing and training in the areas of software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS conducts product testing, builds and improves testing groups and hires testing staff for hundreds of clients worldwide. Ranging from Fortune 20 companies to start-ups, RBCS clients save time and money through improved product development, decreased tech support calls, improved corporate reputation and more. To learn more about RBCS, visit www.rbcs-us.com.

Address:	RBCS, Inc. 31520 Beck Road Bulverde, TX 78163-3911 USA
Phone:	+1 (830) 438-4830
E-mail:	info@rbcs-us.com
Web:	www.rbcs-us.com
Twitter:	@RBCS, @LaikaTestDog
Facebook:	RBCS, Inc.

