Continuous Performance Testing

Managing Application Performance Throughout the Project Lifecycle

CHARIOT SOLUTIONS

Agenda

- Introduction
- 9 Steps to Continuous Performance
- Best Practices
- Discussion

Introducing Continuous Performance

CHARIOT SOLUTIONS

Typical Performance Strategy

- Loose requirements defined:
  - Provide “industry standard” response times
  - Load estimates per day or per hour, not concurrent or peak, often out of date
- Functionality developed
- Fingers crossed
- Performance testing begins

Impact of This Approach

- Performance becomes a major risk factor
- Early design flaws are embedded by future application development
- No performance visibility during development
- Pressure to “hack now, tune later”
- Code and decisions have been forgotten by the time performance issues point back to them

Late fixes are expensive

Worst-Case Scenario

- Deadlines are missed
- Budget is overrun
- Final release is flawed:
  - Poor latency = poor user adoption / productivity
  - Poor scalability = overloaded during peak usage, most frustrating when it would be most valuable
- Project team has moved on
  - Next team: “application is easier to rewrite than fix”
- Reputations of application, team and company suffer

Case Study

B2C Web Application

- Changing requirements led to rework
- Used up slack in schedule
- Developers raced to implement complete business functionality by deadline
- Performance ended up on the back burner
  - "Let's get it working first!"
  - "We can always throw more hardware at it later."
  - Not recognized as a critical success factor

Case Study

B2C Web Application – Result

- Delivered application fulfilled all business requirements, but was not usable
  - Front page took too long to load
  - Many searches caused system to hang
  - Could not scale to more than 20 users
- Subsequent teams went on to spend 200% of original budget trying to fix performance; end result was still sub-par
- Users would have preferred a solid release that only implemented half the functionality!

Case Study

B2C Web Application – Selected Issues

- Analysis
  - No performance/scalability goals identified up front
- Design & Tool Selection:
  - Data and object models were too complex
  - Individual portlets made too many network calls
  - Tool-generated configurations were too verbose and hurt framework performance
- Implementation:
  - Improper use of persistence frameworks
  - Non-scalable code & stored procedures never noticed

Case Study

B2C Web Application – Lessons

- Establish performance goals up front
- Don’t assume that users value feature completeness over performance
- Use prototypes to validate performance of new technologies and designs
- Test application performance early
- Detect and fix design problems
- Correct dev problems before they multiply

Why do we still do it that way?

- Difficult to test performance early
  - Unfinished applications
  - Integration points
- Tools are limited
  - Expensive (often rented)
  - Require specialists
- Don’t want to waste time optimizing early

What if…

- Applications were built with scaffolding so they could be tested right away
- Every developer had access to performance tools
- Load tests were part of the daily build cycle
- An email alert told you that your latest check-in reduced lines of code, but tripled one particular page's response time

9 Steps to Continuous Performance

CHARIOT SOLUTIONS

9 Steps Overview

1. Make a commitment
2. Prepare your organization
3. Prepare your infrastructure
4. Gather and analyze requirements
5. Architecture & Design
6. First Implementation Slice
7. Iterative Development
8. Testing and Monitoring
9. Delivery and First 30 Days

But first…

- Do the costs of Continuous Performance testing justify the benefits?
  - Is it a mission-critical application?
  - Is there a high peak load or a large number of users (Internet applications)?
  - Do automated inputs or feeds cause load?
  - Do competitors offer performance or scalability numbers?
- It's not for every project

1) Make a Commitment

- What's your track record for performance?
- Can you put that in dollars?
- Do you have visibility into current projects?
- If performance has been a problem, then:
  - Get high-level commitment
    - Performance as first class priority
    - Budget for time, tools and equipment
  - Decide how you'll know if you've succeeded

What's Your Performance Track Record?

- Analyze past projects:
  - Project schedules
  - Performance-related bug metrics
  - Employee time records
  - Performance consulting costs
- Look at hardware and licensing costs for vertical and horizontal scaling

How Do You Track It Going Forward?

- Get serious about project metrics
- Store performance test results on an ongoing basis
- Monitor current state compared to defined performance goals
- Have automated tests report bugs for serious problems

Step 1 – Checklist

- ❑ Application performance has been a problem
- ❑ There is high-level support for fixing that
- ❑ Budget has been discussed and allocated for tools and training
- ❑ Project managers feel comfortable scheduling early performance work
- ❑ There is agreement on how to track and evaluate success of new performance methodology

2) Prepare Your Organization

- Train architects and developers
- Investigate tools and frameworks
- Research and establish SLA goals
- Make budget available to projects for performance tools and equipment
- Equip every developer with profiler, load testing and monitoring tools
- Invest in project automation to make continuous performance testing easy

Case Study

In-House Development Team

- Mercury tools limited to a select testing team, run overnight in small window in one environment, very limited availability compared to project need
- Missed deadlines or outages could lead to major loss of revenue and/or reputation
- Even on projects where performance was critical to business success:
  - Performance was not tested until the end
  - In the weeks before key deadlines, dozens of people worked overtime to test, tune and push new releases

Case Study

In-House Development Team - Solution

- Project switched to Apache JMeter, an open source load generation tool
- Tests run continuously on workstations and small servers during development
- At key milestones, tests run on production hardware
- Results in both environments used to find and address performance issues before crunch week
- New tools and new skills led to new best practices and significant improvement

TIPS FROM THE TRENCHES

What does every architect need to know?

- How to gather performance requirements
- How to account for distributed components
- How to build a rapid prototype to test the performance of a new design or technology
- How to plan an implementation so that performance can be tested early
- How to decide what needs immediate attention and what can be safely deferred

TIPS FROM THE TRENCHES

What does every developer need to know?

- How to write or record performance test scripts
- How to run a load test
- How to recognize problems caused by server or test configuration rather than code
- How to recognize database performance problems
- How to use a profiler to analyze a performance problem in code

Step 2 – Checklist

- ❑ Most architects and developers have been through some performance training
- ❑ Tools have been evaluated, selected and distributed
- ❑ Internal documents describe typical performance requirements and SLA guidelines
- ❑ A project automation system and test hardware are available to help project teams run nightly load tests
- ❑ Everyone can check out and test a simple application to see how it works

3) Prepare Your Infrastructure

- Gather and document performance characteristics for:
  - Existing applications
  - Common integration points
  - Common products and technologies
- Provide test environments for every production system

Case Study

Very Large Financial Trading System

- Rather than re-implement functionality, team reused existing services
- No performance profiles were available
- Resulting chain of services couldn't come close to meeting performance requirements
- Not discovered until application was in production

Case Study

Financial Trading System – Lessons

- Plan for reuse by documenting performance characteristics of reusable services
  - Scalability
  - Minimum and typical response times
  - Peak vs. non-peak load
- Don't just assume that a service can be scaled – it may have its own dependencies
- Test your assumptions early and often

How Can You Test Integration Points?

- Ideally, test environments should exist for each integrated system
- Life is not always ideal
- If you know the performance characteristics, then:
  - During development, mock up interfaces and services to mimic those characteristics
  - Test that your implementation can meet *its* targets
  - Of course, find a way to do true integration tests before going live (using spare failover hardware if possible)

Step 3 – Checklist

- ❑ Performance characteristics of all systems are documented and understood
- ❑ Test environment supports load testing against all integrated systems (either via separate test environments or mock-ups)

4) Gather Requirements

- Establish project-specific performance goals
- Understand dependencies on other systems
- Ensure that goals are realistic
- Agree with stakeholders on prioritization of:
  - feature completeness
  - throughput
  - user perception (e.g. time until first response byte)
  - response time
  - scalability to handle future growth

Perception is Everything

- Typical web response time requirement is 1-2 seconds
- Your application may need to contact several other systems to complete a user transaction
- Before spending to upgrade infrastructure, decide if it's acceptable to:
  - Immediately direct user to progress animation
  - Kick off transaction on server
  - Redirect user once transaction is complete

Usage Patterns Matter

- The marketing team for your consumer-facing web application puts together an email campaign
- What will happen to your system if they send out an attractive offer to 500,000 customers on Monday morning?
- What if they send it 10,000/hour throughout the week?
- It's important to know what will drive usage and what level of warning/control administrators will have

Step 4 – Checklist

- ❑ Performance requirements and priorities for project are well understood
- ❑ Dependencies on other systems are well understood
- ❑ Typical and peak usage scenarios have been documented
- ❑ Future growth has been estimated and accounted for

5) Architecture and Design

- Analyze requirements and dependencies
- Plan communication patterns between services and set performance goals accordingly
- Use prototypes to test new designs, e.g.
  - First time using a portal architecture
  - First time relying on web services
  - Performance-critical sections of application
- Use strategies that allow for later optimization:
  - Where possible, avoid clustering pitfalls in code
  - Encapsulate data access to allow for later caching

Case Study

Two Application Development Teams

- Requirements called for application to handle 200 concurrent requests with <2 second response time
- Development work was split between front-end and back-end teams
- Stubs were used to mock up interfaces
- Back-end team met their performance goals, but front end used unexpected call sequences and did not separately test front-to-back performance

Case Study

Two Development Teams – Lessons

- "Unit test" performance is not the whole story
- Coordination between teams is critical
  - Integrate early to discover unexpected issues
- Front-to-back testing is the only way to accurately model the end-user experience

Create a Design Portfolio

- Document architecture and test results for multiple internal systems
- When working with new technologies, create reference architectures with known performance characteristics
- Create and test a realistic proof of concept for popular new technologies *before* committing to them

Step 5 – Checklist

- ❑ Performance goals have been set for each service and/or tier
- ❑ New technologies have been evaluated and tested with prototypes
- ❑ Design allows options for future optimization without getting too abstract

6) First Implementation Slice

- Ramp up project automation tools
- Plan first iteration to exercise all parts of the design
- Use scaffolding techniques to quickly generate testable code
- Begin generating or recording test scripts
- Have architect monitoring test results aggressively to watch for design problems
- Be prepared to switch technologies or revisit SLA requirements

Case Study

Customer Relationship Application

- During analysis and design phases, detailed web flow diagrams were created in Visio
- Stakeholders signed off on this UI design
- UI designers then expanded the diagrams to show how Struts actions and JSPs would interact
- Design looked good. What next?

Case Study

CRM Application – Scaffolding in Action

- UI Designers created headers, footers, stylesheets
- Exported Visio diagrams to XML
- Developer wrote script (in half a day) to generate classes, JSPs and tests based on Visio XML
- With the push of a button, Visio diagrams were translated into completely navigable web app
- UI Designers walked users through app to validate design; changed and regenerated it a few times
- Developers implemented pages one by one

Case Study

CRM Application – Scaffolding Results

- Rapid turnaround between page flow modeling and first application walkthrough
- Tests could be written/recorded and run during development
- Accelerated UI implementation
- Future projects used similar techniques to generate scaffolding across all tiers

TIPS FROM THE TRENCHES

All tests pass! ...or do they?

- Many applications don't return proper error codes
- Load testing tools see everything as green
- Looks like you're scaling to the moon
- If you turn on response tracking, you can see that most results are app-generated error pages
- What to do?
  - Fix application to return proper codes (e.g. 400, 500)
  - Set up rules in tool to detect application error pages
  - Set up tests that look for specific fields in response

Step 6 – Checklist

- ❑ All major design aspects have been exercised and tested
- ❑ Test results show that performance is near or within desired SLAs
- ❑ All developers and test engineers are comfortable writing/recording performance tests
- ❑ Automated test runs and alerts are working properly

7) Iterative Development

- Ensure that nightly tests are run on realistic hardware
- Configure tools to generate alerts and file bugs for serious errors or regressions
- Deal with serious performance issues by the end of each iteration (as with serious functional problems)
- Have architect monitor test results and decide what is serious

Project Tool Integration

- If you stick with open source tools and commercial tools that have open APIs…
  - Bug Tracking (JIRA, Bugzilla…)
  - Source Control (Subversion, CVS…)
  - Builds (Ant, Maven, Scripting languages)
  - Tests (JUnit, JMeter…)
- Then you can easily integrate them
- Imagine… test fails, developer gets email, reads bug report, fixes bug, checks it in with bug number in commit message, test gets rerun, bug gets closed

Step 7 – Checklist

- ❑ Nightly performance tests run consistently
- ❑ Broken tests are considered a build failure
- ❑ Results are monitored on a regular basis
- ❑ Major performance issues are resolved by the end of each iteration

8) Testing and Monitoring

- Performance should already meet requirements
- Keep automated tests going to ensure that bug fixes and “cleanup” don’t change that
- Test thoroughly on target hardware and software
- Implement monitoring strategy

Don't Wait for the Second Lightning Strike

- Often, we don't start monitoring applications until after a problem has occurred
- Intermittent slowdowns are often network related
- You'll need comprehensive monitoring to prove it
- Test what happens when your application is overloaded and ensure that you'll be able to:
  - Quickly recover
  - Log enough information to reproduce it in a test lab
- Monitor response times and generate alerts so that investigations can start before users complain

Step 8 – Checklist

- ❑ Application can be monitored remotely
- ❑ Performance continues to meet goals
- ❑ Performance has been thoroughly tested on production hardware
- ❑ Integration has been fully tested for performance

9) Delivery and First 30 Days

- Transition monitoring and testing tools over to production support team
- Document performance characteristics and limitations
- Document design considerations
- Start capturing real traffic and using it to create more realistic load tests

Best Practices for Continuous Performance

CHARIOT SOLUTIONS

Best Fit

- Mission-critical projects with serious performance or scalability requirements
- Any project where performance is the top priority
- Projects at organizations that have had problems with performance in the past
- Projects using new technologies and architectures

The Script Monkey

- Have someone on the team or nearby who is great at scripting and gluing tools together
- Make it easy to record and run tests
- Make it easy to find and interpret test results
- Will need some time up front, but the results can be reused across projects

Typical Setup

- Application designed and scaffolded
  - Including login, accurate navigation, test data, etc.
- Test scripts recorded based on use cases
- Test scripts saved and checked into repository
- Performance tests run automatically (cron, etc.)
- Results saved in structured format for subsequent parsing
- Reports compare and interpret data, show what's changed since the previous run, send bugs or alerts as appropriate

Designing Performance Tests

- Low-level tests aren't as useful
  - Real performance depends on end-to-end processing
  - Test environment doesn't reflect runtime environment
- Best to base on use cases
- Emphasize paths that are likely to occur and are sensitive to performance

Designing Tests, cont.

- Real users can record test scripts for you as they use or explore application
- Don't forget external triggers like email campaigns, advertisements, Slashdot…
- Don't forget feeds, direct APIs, web services, and other system interfaces
- Plan to update as necessary going forward

Monitoring Test Results

- When implementing a new feature, a developer can run a load test to see how it performs
- Architect or lead developer should look over performance runs once a week
- Test errors should be fixed ASAP
- Clear up major performance problems by the end of the iteration
- Record major events to cross-reference with test results (server configuration changes, caching…)
- Towards end of project, use alerts more heavily

Resolving Test Errors

- May be caused by:
  - Product configuration (memory, pools, etc.)
  - Application changes (hyperlink changed to form submission, page flow changed)
  - Data changes (search terms no longer valid)
  - Code errors (concurrency problems, scalability problems, bugs)
  - Outages (planned maintenance, etc.)
- Best to resolve quickly, because it taints performance data

Desired Outcome

- “Code” should run quickly
- Background steps (looking up data on current user, populating drop-downs) should run quickly
- Major delays should be in dealing with external systems (database, messaging, legacy, etc.)

If not…

- Run tests manually and review progress in real time
- Compare results of one test alone to one test with other competing requests
- Review individual steps within larger tests
- Monitor CPU, I/O, database CPU, etc.
- Follow up with profilers and other appropriate tools

Tips For Better Tests

- Reduce log output
- Pay attention to server configuration
- Warm up the server/JVM before recording test results
- Use ramp-up to avoid all users hitting the same feature at the same time
- Spread threads across user accounts, search criteria, etc.

Types of Tools to Consider

- Test suites
- Load generators
- Test run automation tools
- Profilers
- Runtime monitoring tools
- Platform-specific tools
  - Database
  - Application Server
  - Interpreter, VM, Runtime Environment
  - Operating System
  - Network

Discussion

Erin Mulder: emulder@chariotsolutions.com

Aaron Mulder: ammulder@chariotsolutions.com

Download slides at: http://www.chariotsolutions.com/

CHARIOT SOLUTIONS