DrupalCon Session: Making Sense of the Numbers: DOs and DON'Ts of Quality Performance Testing

Presented by @erikwebb and @doogiemac | Session Information

Everyone Needs Performance Testing

Performance testing isn't just for sys admins or developers or the final launch date; it's affected by everyone's decisions throughout the project life cycle. Everyone needs to buy into performance.

Background Information: Getting on the Same Page

- **Performance** is about more than just how long it takes to load a page from the website.
    - It's also about how long it takes to render, how long it takes images to load, how long JavaScript assets take to load.
- **Scalability** is about what happens when one user hits your site vs. when thousands of users hit your site; what does it do to the load time?
- **Scaling** is how you add servers to improve the performance of your site. Horizontal scaling is adding more nodes to a system (e.g. servers). Vertical scaling is adding more resources to a single node (e.g. improvements to an existing server).
- **Load testing** is measuring the impact of an increased number of users; it's not about breaking the site, unlike...
- **Stress testing** is seeing what you have to do to your server to make it crash and burn
- **Contention** is a fight over resources and is one of the most important concepts to understand when thinking about performance. Think about Wal-Mart on Black Friday -- it's LOTS of people trying to fit in a door that only fits a few people at once.

Do Have a Plan

- Create representative user scenarios
- Script the actions of the simulated users
- Analyze the results of the test

Do Involve Everyone

- Involve developers
- Involve sysadmins, developers, and project managers
- Involve project stakeholders
- Remember that **everyone** is the QA team!
- Align tests with business goals

Do Set Technical Goals for Testing

"If your tests aren't good, the numbers you get from them are irrelevant. If your tests are fantastic, you can take them and decide how much more money you need to spend on servers or other resources."

- Monitor changes and usage side-by-side
- Ensure new development does not affect existing functionality
- Reduce infrastructure complexity through early testing
    - If you don't do performance testing early in the project, it can be next to impossible to identify why your site is slow -- there are so many factors.
    - If you constantly monitor performance and check in as you add features to your site, you set yourself up for a much higher chance of success.

    - When you build a feature, like an article content type, don't just make a few of them and check to see how long it takes to load -- make a LOT of them, comparable to the amount you'll ultimately have, and then test.
- Hold programmers accountable for performance, not just sysadmins

Do Set Specific Success Metrics

- Backend performance
    - X concurrent authenticated sessions
    - X page views per minute
    - X seconds maximum per request
    - X MB memory usage per request
    - Maximum % CPU or GB RAM used on server
- Frontend performance
    - X seconds until initial render
    - X seconds until full page load

Figuring out what these metrics are is project-dependent and client-dependent. These are benchmarks so that all people on the team can check in throughout the project to find out whether they're succeeding -- it's important to get them right.

Do Gather All the Data

- Never trust the client
- Find the most value for the client
- Understand the full client use cases
    - Solve the business case first. If saving an article takes too long, but your primary requirement is to get anonymous users the ability to view each page in a few seconds, keep your focus on the primary goal.
    - ...but don't disregard obvious problems.
- Historical data is a **requirement**
- We're too smart to ignore our intuition!
    - If you feel like there's a bigger problem the numbers aren't revealing, don't ignore that feeling.

Don't Give Up Until It's Done

- Get your hands dirty
- Don't get lost in Drupal - don't assume that Drupal is being slow when it could also be an external web service, a problem with a cookie that's throwing off Varnish, etc.
    - Anecdote: A client had a site with several hundred thousand pages and 5,000 taxonomy terms. They had a very complex query that was taking a long time to return results. They souped up their database and did a lot of "performance improvements," only to find out that the query itself was only taking 1 second -- it

was taking 4-5 seconds to render the results in Drupal. Make sure you're troubleshooting the right problem.

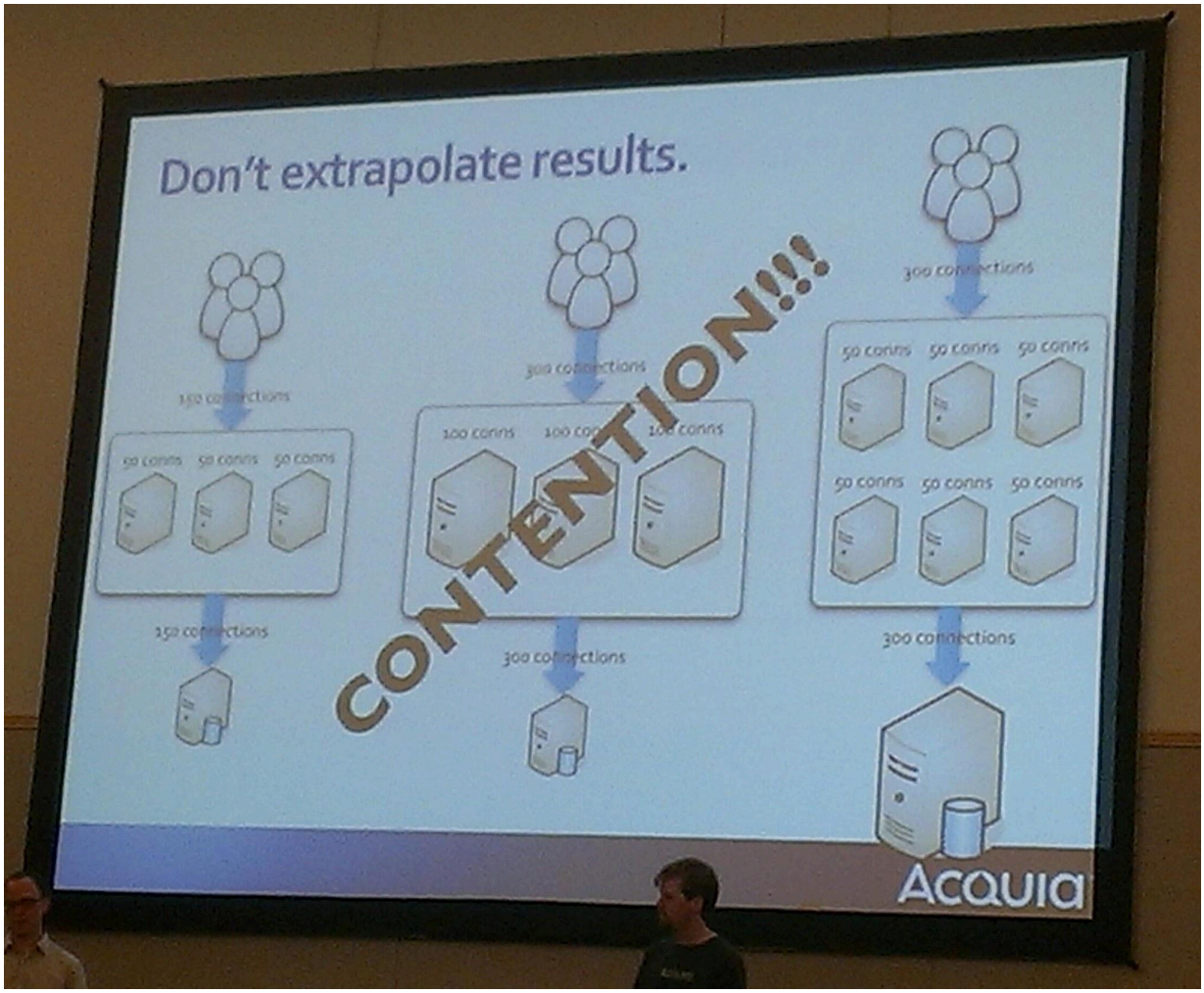
- Assume no one has checked the easy stuff yet
- Don't accept anything less than perfect
    - Re: the article saving vs. 2-5 second page loading example above, prioritization is good -- but don't ignore obvious problems just because they're not in the initial requirements. If things are bad, fix them.

Don't Use a Dev Environment for Testing

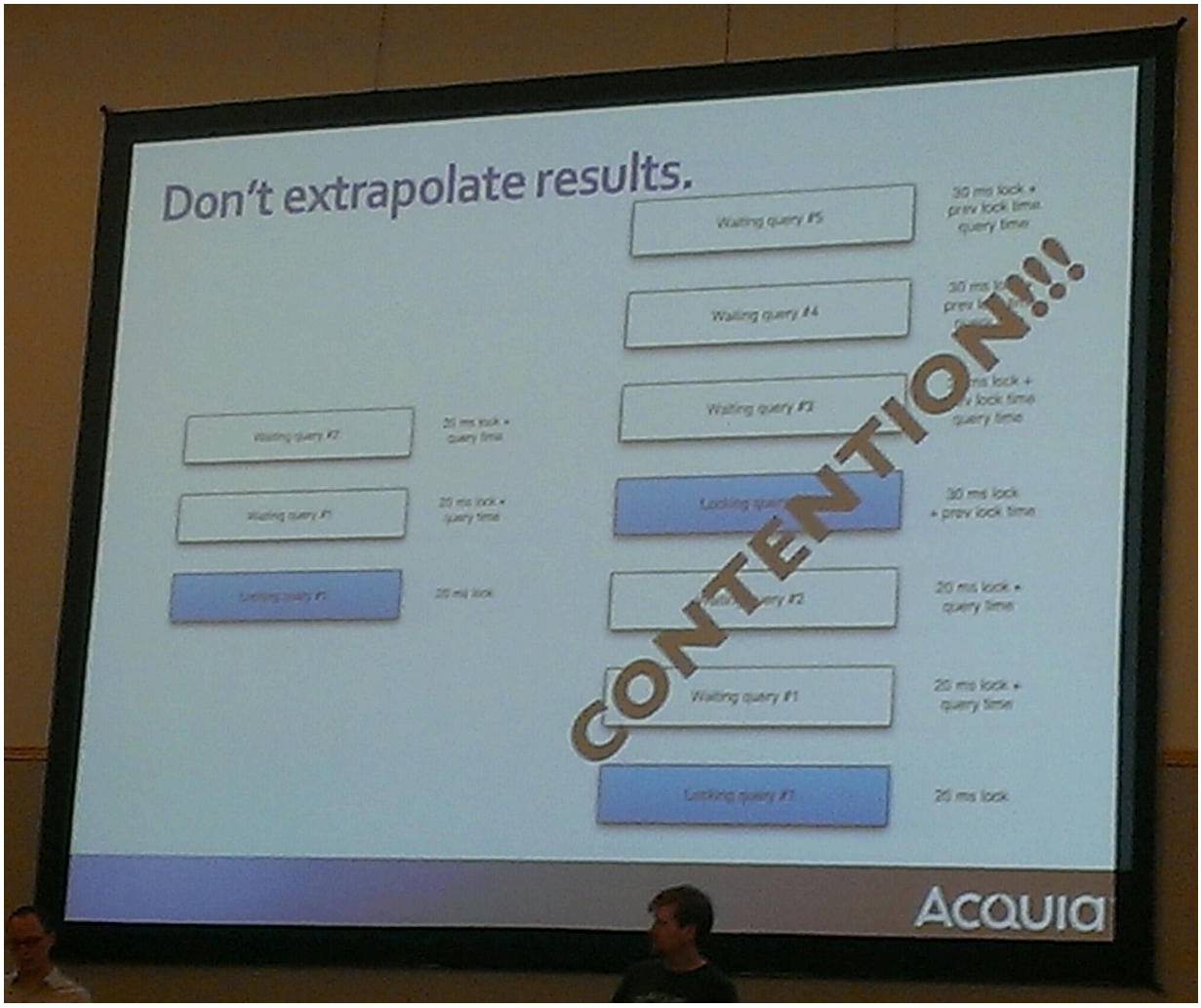
Your numbers are only as consistent as your infrastructure.

- Developer overhead - XDebug, XHProf, etc. - interferes with your testing because it's not consistent with prod
- Verbose logging is great for development, but not performance
- Congested networks for dev sites won't give you a realistic view of your performance

Don't Extrapolate Results



In the example above, you might assume that because you were okay with 150 connections with your three nodes, you could double the resources of each node to support 300 connections. But you might actually need to double the number of nodes. Don't guess; test.



In the example above, you might know there is a locking query that costs 20 seconds. But remember that in a production environment, there might be multiple locking queries stacking on top of each other. For example, if the cache is cleared, the menu cache will have to be rebuilt -- but so will other caches. When all of these individual locking queries you may have written off hit at the same time, they add up to a much more serious problem.

Do Write Smart Tests

Your numbers are only as good as your most current test.

- User scenario created by business
- Use real data from analytics
- Ensure variability
- Measure test coverage
- New tests for each feature

Don’t Write Your Tests for Launch

- Launch is a milestone, not the end
- New features are added all the time, and you can have dramatically unintended consequences when you add them
- It’s just QA!
    - Launching is about meeting the scale of the site when it goes live. It shouldn’t mean that your test patterns change.

Do Use Continuous Integration (CI) for Performance Too

- Ongoing content scaling complicates long-term reliability
- Integrate with Jenkins (or other CI tool) for performance regression testing
    - Seeing a performance trend graph enables you to go back and find out what was committed when performance changed.

Do Multiple Types of Testing

Your numbers are only as good as your tools.

- Request profiling
- Service testing
    - Test each level of the stack to avoid false assumptions about your one, final number
- Simple HTTP response testing
- Load testing

Virtual vs. Real Load Testing

Very different, very different costs. Virtual is full of shortcuts. Real provides real data, as experienced by users, using a mouse, keyboard, etc.

- Virtual
    - HTTP client
    - Designed for efficiency
    - Limited client functionality
    - Example: JMeter cloud service - $500/**month** up to 4800 concurrent
- Real
    - Browser client
    - Estimates real user experience
    - Supports AJAX natively
    - Example: Selenium-based service - $499/**week** up to 100 real users

Front-end performance is better tracked by real load testing.