Mindreef

White Paper

Agile Load Checking for SOA Quality

Breaking Performance Barriers when Load Testing Service-Oriented Architectures

The Problem with Load Testing

Performance testing business applications has traditionally been the domain of performance and testing experts and often occurs late in the development cycle. The tools commonly used are expensive, hardware-intensive, and complex to setup and use. The people who can make the most effective use of the technical tools are not the ones who best know the business transactions. With enough brute force effort, translation from one domain to another can happen, but only periodically and very likely not at the pace of change required in an agile organization. This contributes to the ongoing disconnect of IT from business, a challenge SOA is expected to help fix.

Without addressing common barriers to improved performance testing, services and consuming applications will fail to meet required service levels and user expectations. By aligning performance testing practices with the goals of the business, SOA projects can avoid common performance problems and realize more of their potential.

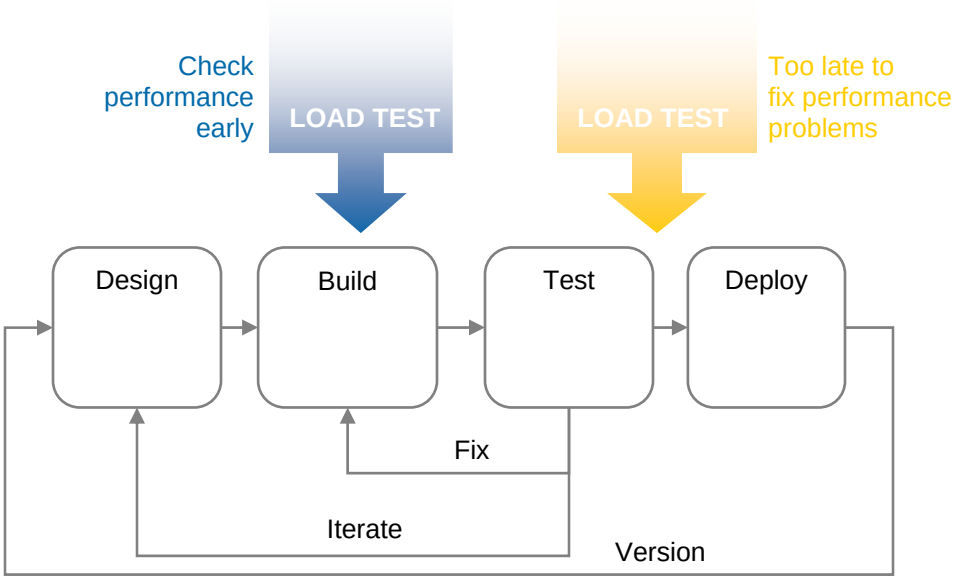
SOA Load Testing Barriers

- Load testing is typically done late in the development process; performance and scaling limitations are "baked in" and difficult to fix
- Many pre-deployment load tests on Web services fail, causing project delays as performance issues are corrected
- Load testing is typically the responsibility of domain experts (an individual or group) separate from the QA and dev process
- Load testing tools are typically complex and expensive software
- Most traditional tools apply load as virtual users; Web services often have different interaction patterns.
- Traditional load testing is done in a lab environment, a SOA application will rely on external services already in production.
  - Can't load test if a dependent service belongs to a third-party
  - Can't load test if a service is a "pay-per-use transaction"
- Many services not deemed performance-critical are not load checked, causing future problems as they get re-used and become strategic

Fundamental Barriers to SOA Performance

Testing too Late to Fix Problems

Has this happened to you or someone you know? A team works on an application or service on a tight schedule, building and refactoring functionality based on insufficient and/or evolving requirements. Following some number of coding, testing, and bug fixing cycles the team is ominously close to the end of the schedule. With the application nearing completion the team earnestly begins performance testing, without knowing what to expect for response times or scalability characteristics,. After struggling with learning how to use load testing tools, the team gets its first results, and they are not encouraging. They realize that instead of sub-second response times they will be lucky to have sub-minute responses, and what's worse with 5 concurrent requests, the system locks up and becomes completely unresponsive.



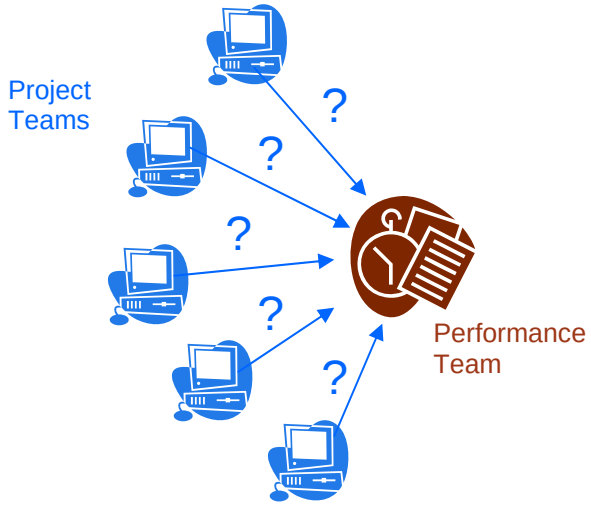
Testing early and often is a mantra in modern software development shops but load testing is often excluded. The cost and complexity of load testing tools and the focus on getting software to work can make early load testing seem impractical. But

performance and scaling limits of a system are often deeply baked-in as a system is developed. Database schema designs, transaction scopes and semantics, thread contention, locking patterns, and third-party components or dependent services are among common sources of baked-in performance and scalability limits. These early design decisions are not easily tuned at deployment time but are easy to uncover with even modest load testing during development.

Some people misapply the "premature optimization is the root of all evil." quote by Sir Tony Hoare and popularized by Donald Knuth [1]. This often quoted and misapplied bit of wisdom referred to counting cycles and assembly language statements instead of structuring code in a readable and maintainable fashion. Continuously load checking Web services early in development (where optimization is not the goal) provides inexpensive oversight of the evolving performance characteristics of a service or application. By making load checking a part of the continuous integration system, the development and test teams can watch trends in key performance metrics over time. If a change seriously impacts performance of the system, that code change can be questioned immediately and an alternative approach investigated. Contrast this with the current approach: code until the software works, then find the performance bottlenecks in the completed system after you've used all your allotted time for developing the system.

Inefficient Use of Performance Testing Teams

In environments where performance is important, you commonly find shared performance testing teams of experts equipped with large, expensive, and complex load testing labs. These teams are shared with many project teams and can be a bottleneck if not managed properly.



Experienced and skilled performance experts are invaluable in tuning system runtime configurations, anticipating scaling bottlenecks, and estimating production hardware requirements. But they can't provide this valuable information if they are testing applications and services that are not ready to test - the *status quo* in many organizations.

All too often, applications and services are load tested for the first time by shared performance testing teams. This team of experts can spend much of its time

tweaking log file settings, adjusting connection pool sizes, externalizing configuration parameters, creating database indexes and other tedious and predictable tasks that the project teams are more capable and qualified to do. Studies show that organizations get much more benefit out of making these adjustments early in the development cycle and testing them on small-scale requests, such as 5 or 50. SOAPscope Server and SOAPscope Workstation are especially useful for performing these types of adjustments to your load testing.

Getting an application ready to load test can greatly reduce the time the performance team needs to spend on mundane aspects of test preparation so they can apply their skills where they matter most. It also prevents embarrassing situations when performance teams need to return to the project team because the system is just not ready or never will be ready to scale.

As organizations transition from traditional siloed application development to services and composite applications, the projects will be smaller, quicker and more numerous. This puts pressure on shared performance teams used to spending much more time with a system. Instead, they will need to test more services and applications in a shorter period of time then they are used to. To avoid being a bottleneck, the services and applications need to begin in a better initial state – ready to test – so teams aren't wasting their time doing the work better suited to the project teams.

Testing Not Done from the Consumer's Point of View

Testing performance at the right place is important; a widely accepted tenet is that systems built for others to use should be tested for performance in the consumer's interface. For SOA, performance testing should occur in the service interface. Respecting the service interface boundary and the SOA tenets [2] allows load testing to black box the service implementation – masking complexity and providing a relevant and unambiguous place to measure.

Reusing unit testing frameworks for performance has merit if your service has a particularly performance intensive area of code or database interaction but unless you consider your consumer's point of view, you will not have an accurate picture of the real performance issues. These include the application hosting environment, protocol marshaling layers, network of intermediaries and geographical considerations.

As service-oriented ecosystems become more mature, performance issues will focus less on the individual service and more on the business process being automated. It might have been acceptable in the past to determine that an individual service could handle thousands of requests per minute, but if that service is combined with ten other services in a process flow – it's the entire process flow that is important, not just the one service. In this case, the rate limiting step could be a service that was not considered performance critical when it was first deployed. Then, other services included the service in subsequent development, but no one ever revisited the service to determine if it's affect on performance changed. Performance testing at the scenario level above individual requests and response to a service is key to eliminating bottlenecks across the overall architecture.

Prohibitive Cost Model of Tools

Most existing performance testing products are priced based on the volume of load created. In a web environment, that maps to *virtual users*. Traditional load testing is priced this way because of the strong correlation of web site traffic volume to the cost of the web site decreasing. That is, the revenue potential for a website is often directly proportional to the number of visitors. The more visitors to a site purchasing books, flowers, or antique clocks, the more direct revenue realized by the web site owner. The same value mapping exists for enterprise applications where broad use implies value. When the central timesheet application goes down, the entire enterprise is negatively affected. Though SOA and specifically, Web services are based on the same underlying technologies, the economics are completely different.

Unlike with web sites, the consumer of a Web service is another software program, not a person using a web browser. Often the consumer is another service connected in an event-driven architecture implementing a business process. The types of consumers and interaction styles between them are very different from the scenario of a person visiting a website. It's possible that one service consumer can create more transaction volume than 100 users with browsers because of the implicit human *think time* involved with people. Web site visitors click links once per 3 - 10 seconds while a consumer of a Web service might be sending requests over a SOA 100 times per second. This makes the interaction profile for SOA services so different.

The request/response models between web applications and SOA applications are also different. Web pages are designed to make small requests and handle larger responses – click on a link and return data. In SOA applications, a SOAP message request might contain much more data can than its response. For example, a service consumer might upload a large amount of database information and only receive a confirmation back from the application.

The testing model also needs to change for SOA applications. Companies with traditional products based on the web application approach will find it difficult to adjust their testing model. Traditional load testing is "hardwired" around certain numbers: less than 1 second page load times with the following ranges of users: 100, 500, 5000 virtual users. Testing with 20,000 virtual users makes sense for some large web sites but those numbers aren't applicable to Web services.

Load testing tools must recalibrate to the specifics of Web services technologies and their new systems performance characteristics. Fifty simultaneous requests might be enough concurrency to test as the actual transaction rate might be much higher and more complex then a simple HTTP web form posting. This makes the "virtual user" based price model inappropriate for Web services applications. If you buy a solution based on the traditional web application approach, you are likely to pay too much.

Although there are several free or open source load testing tools and framework available for your early load testing, these can be expensive in both time and money if you incur significant internal development and ongoing maintenance costs to make them work effectively for you. Be wary of buying a load testing solution that only your most capable developers can use.

Limited access discourages Regular Performance Tests

Performance testing experts encourage load testing early in the development cycle for good reasons [3] but the technical complexity of typical tools and pricing structures prevent many organizations from following this recommendation.

Most businesses cannot justify investing hundreds of thousands of dollars in software, hardware and people to run a load testing capability for every project that would benefit. When a performance testing capability does exist, it is often shared across many projects to amortize cost and share a scarce skill set.

The extent of the performance testing "skill siloing" varies across organizations but it limits the potential benefits of load testing. At one extreme is a dedicated performance test lab with a skilled performance test team. A variation on that theme is a vendor supplied test lab as part of a partnering arrangement - an extreme example of the "skill silo". The other end of the extreme is the dedicated tester with a single copy of a load testing tool. In either case there is a disconnect between performance testing capability and the rest of the team.

These disconnects limit the effectiveness of a load testing capability:

1. Load tests aren't performed early enough to expose fundamental architectural issues affecting performance and/or scalability. This can result in "baked-in" performance limits that can't change late in the process, due to the rework required.

2. Load tests aren't even performed on all projects. Judgment calls are made based on cost and availability. These judgment calls are much easier with applications than with services. For example, a service that was not deemed strategic today might eventually be very strategic in the future, depending on the processes that build on it.

3. Performance testing and goals are viewed as "voodoo", outside the domain of the development team building the services. Often the development team will not have any accurate estimates on the anticipated performance and scaling characteristics of a planned service. As a result, questions about these metrics are met with blank stares, guilt, and frustration.

The ideal solution is to extend the existing development and test environment to include performance testing. This puts the tools in the hands of the people that can do the most good at the right time. This implies a major change in approachability as well as price.

The Separate Lifecycles of Service Providers, Consumers, and Metadata

Most load testing tools used for Web services test an implementation. The traditional, waterfall approach to application design, with load testing before deployment does not apply to SOA because changes to intermediary metadata, virtualization, ESB configuration, routing, transformation and security policy modification all significantly impact service performance. Testing to ensure Service-Level Agreements (SLAs) are still possible after changing your metadata requires

testing after a service has been deployed – perhaps, when the service development team no longer exists and cannot retest.

Design-time and run-time do not apply to SOA, because services are *always* in run-time. The traditional software development lifecycle (SDLC) has evolved into separate lifecycles for service providers, service consumers as well as policy and intermediary metadata that require continuous testing. These lifecycles are often separated by both time and distance. The impact of traditional load testing on a remote, dependent service used in a production application can be catastrophic to a business.

Agile Load Testing

SOA enables agile testing when testing is focused on the architecture and XML abstraction layer versus the code implementation. By testing the XML abstraction layer and using the interface and contract standards, you can not only test first, but test continuously in the SOA and validate interoperability for service consumers. The ideal solution for ensuring SOA performance levels is to provide a direct extension of the existing development/test environment to address performance testing early and often based on the consumer's perspective.

Today's agile, test-driven development processes are becoming more and more common for SOA projects. This requires putting new tools in the hands of the people who can do the most good from start to finish: the development teams. This also means a major change in the approach and pricing model for load testing Web services in a SOA. Traditional load testing was based on the user interface of an application, simulating a user clicking on links. But the UI is typically the last thing finished in an application, which essentially prevents early load testing. With SOA and contracts (WSDL) being developed first, load testing can be done from the contract point of view. This allows load testing to occur at the beginning of the process after the contract is written. Combining the premise of agile development's "test-first" approach, with SOA's contract-first approach enables "Load-Driven" SOA development.

SOA supports and encourages constant change to provide and improve business agility. The traditional load testing approach and technologies used for load testing hinder change. Final-step load tests that fail often, potentially impact production and delay application deployments are detrimental to the core value of a SOA to a business. The agile load testing approach supports the continuous change of SOA – preventing a SOA from becoming static and rigid.

Benefits of Agile Load Testing

- Quickly and easily test and validate daily builds, early in the development cycle, when it is easier / cheaper to fix problems
- Avoid project delays and poor performance due to last minute testing, when it more expensive to fix problems, if they even can be fixed
- Eliminate the dependencies on complex costly test tools designed for legacy applications, the need for to test labs to perform the testing, and the need to restrict testing to test specialists

- Improve communication and share performance testing validation with non-technical staff to increase trust in your services

Mindreef Load Check

Mindreef Load Check is an optional, add-on module to SOAPscope Server and SOAPscope Workstation. Mindreef Load Check allows you to easily build scenario tests and add them to a load test, without needing to know XML or scripting. A load test can be up and running in an hour. Mindreef Load Check provides a dashboard that shows the results of each load test run over time. This allows developers to easily detect performance problems when they first occur so they can easily fix them before they get "baked-in" to the code and the team moves onto another project. When there is a problem, Mindreef Load Check provides the detailed information you need to quickly solve the problem. Mindreef Load Check is licensed so the entire team can benefit from the test results. There is a one-time cost for Mindreef Load Check, which makes it available to all users on a SOAPscope Server, with no additional costs for "virtual users".

Conclusion

Agility is a key goal of Service-Orientation. The ability to bring scalable services to an architecture in a predictable and cost-effective way is more important than it was in previous architectures. The best way to accomplish this is to start load testing early in the development cycle where performance problems can be identified early and fixed when they occur. Mindreef SOAPscope Server and SOAPscope Workstation with Load Check provide this solution, enabling organizations to build confidence and trust in the performance, scalability, and quality of their services, as well as the ability to achieve business agility from SOA.

References

1 - The Fallacy of Premature Optimization, Randall Hyde
http://www.acm.org/ubiquity/views/v7i24_fallacy.html

2 – 4 Tenets of SOA, Don Box
http://msdn.microsoft.com/msdnmag/issues/04/01/indigo/default.aspx#S1

3 – Common Testing Mistakes, Brian Marick
http://www.testing.com/writings/classic/mistakes.html