



PACIFIC NW
28TH ANNUAL
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010



ACHIEVING
QUALITY
IN A COMPLEX
ENVIRONMENT

*Conference Paper Excerpt
from the*
CONFERENCE
PROCEEDINGS

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

SIMULATING REAL-WORLD LOAD PATTERNS WHEN PLAYBACK JUST WON'T CUT IT

Wayne Roseberry

wayner@microsoft.com

Test Lead, Microsoft Corporation

ABSTRACT

Load testing is an important part of performance and reliability testing for server software. One of the goals of load testing is to determine as well as possible whether the software will stand up reliably under real-world operating conditions. One way to achieve this is to attempt to simulate in laboratory conditions the traffic and usage patterns that will happen in the real world. The challenge is to create a simulation that models the real-world behavior accurately enough that the results can be trusted as representative and expose product flaws. There are tools available, which provide web-traffic capture and playback, but these tools are lacking when it comes to many complicated client/server protocols and feature scenarios. The goal, then, is to create systems that can capture the data and traffic patterns from a production system, express them as a model, and drive the load test from that model.

This document describes one such tool built by the SharePoint team in Microsoft Office 2010. The document describes the architecture and behavior of the tool, the ways it was applied, the types of product flaws it exposed, as well as limitations of the tool and possible improvements for the future.

BIOGRAPHY

Wayne Roseberry has been in the software industry for twenty-four years, twenty of those at Microsoft Corporation, sixteen years testing software. His product experience includes Microsoft Office, The Microsoft Network (MSN), Microsoft Commercial Internet Server, Site Server, and SharePoint.

1 BACKGROUND

1.1 SHAREPOINT

Microsoft SharePoint server is a web-based collaboration and content publishing application. Customers buy it to satisfy a broad range of business requirements that include the following:

- team communication and issue tracking,
- document management
- simple workflow processing
- enterprise search and information portals
- business application aggregation
- content management and publishing

The newest release of SharePoint in 2010 adds the ability to use web-browser based versions of the Microsoft Office client applications for content viewing and authoring. SharePoint's end user interfaces support web browser, rich GUI client, as well as SOAP and REST based web service programmatic interfaces. The technology stack is built on Internet Information Server, ASP .NET and SQL Server. Further, the Microsoft Office clients, from version XP through 2010 support integration with SharePoint for file storage, workflow integration, and social collaboration. There is a rich API for server side applications, allowing a customer or third parties to extend behaviors to match their business needs.

SharePoint has been in the market since 2001 and has become a strategic part of the Microsoft product offering, with growth rates that outrun any server product in Microsoft's history. Sales of SharePoint to date have been largely enterprise oriented, strongly attached to Microsoft Office client sales. Microsoft likewise offers cloud-based hosting services for SharePoint.

1.2 PREVIOUS TESTING CHALLENGES

Simulated real world load testing of SharePoint is extremely difficult. There is a great deal of complexity and size in the number of different customer scenarios. However, it is likewise difficult to apply a quality bar against tests without mapping expectations against real world patterns. Here are several of the issues that come up:

1.2.1 ABNORMAL VS. TYPICAL USAGE PATTERN DISTINCTIONS ARE HARD TO DETERMINE

It is easy to create load tests that will push the system to its limits and create performance or reliability problems. The difficulty comes in triaging those problems to determine whether they merit a fix in product code. Fixes in SharePoint frequently involve tuning SQL queries, the performance and behavior of which are highly impacted by data and usage patterns. The same fix that works well in one case may be a very bad choice in another case, so it is very important to be aware of the most common usage patterns when selecting performance fixes. This motivates the test team to seek out usage and data patterns that are already validated by production systems.

This becomes difficult with SharePoint because the functionality is broad, and the use cases and scenarios highly varied. In a similar manner, the data sets are frequently large. It is not uncommon for a customer to have multiple terabytes of data in production.

1.2.2 DYNAMIC STATE MAKES SIMPLE PLAYBACK TESTING NON-EFFECTIVE

Capture and playback systems will record requests on a communication channel and then play those requests later on a system under test, usually with a copy of the data that existed on a system at the time

of capture. This works effectively when the communication is purely stateless and predictable. An example would be a web page that always sends the same reply when the client issues a specific request.

The problem occurs when a web page generates dynamic data, or is highly dependent upon a complex state sequence before the request occurs. This forces the client to send requests that preserve whatever the server sent previously, or to reply at a moment when the state is still consistent. In this case, the recorded request is only valid at the moment it was captured. Playback later will be invalid because the request no longer contains the correct information, or did not happen at the moment that the server could process it correctly.

SharePoint has many features that manifest exactly this behavior. For example, when a user adds a new document to the system, that document is given a unique and random identifier that is generated by the server dynamically at runtime. Later pages that perform operations on that file, such as checking it out for edits, viewing or editing properties of the document or launching a workflow against it, pass that unique identifier in the HTTP request to the server. If that identifier does not match one for an existing document, the operation will fail. A generic capture and playback tool cannot anticipate ahead of time what the identifier will be for newly added documents; hence, the test code in this case needs to be more specialized. It needs to anticipate the identifier and remember it for use in later requests.

A similar condition occurs in complex state sequences. For example, SharePoint has a collaboration workflow sequence that can enforce a requirement that users check out a file before editing and then check in that same file before allowing others to edit it. If a user attempts check out on a document already checked out the operation will fail, and similarly if a user attempts check in on a document that is already checked in the operation will fail. The time gap between check out and check in with real world systems is unpredictable. A user may complete the sequence in just a few minutes, or may leave a document checked out or in for days or months. A capture tool, then, runs the risk of capturing a document in the middle of the sequence, and then on repeated replay invalidates the sequence. The mitigation is not to use generic tools, but again to write test automation that is aware of the state of the system and apply the tests properly according to the rules of the system (or at least forcing the sequences you desire to test rather than having them break because the capture system is not flexible enough).

1.2.3 DATA SAMPLES ARE CRITICAL TO PERFORMANCE AND RELIABILITY

SharePoint data sets are large and complex. There are many databases in a SharePoint deployment, and many different types of objects that can grow in size per object and in quantity of those objects. For example, a document library may contain files that are very large or very small, and likewise can contain just a few documents, or perhaps millions of documents. These two different variables for document libraries combine together to make the actual performance characteristics more complex than if it were just a matter of size of file or number of files. Just as usage patterns will push the database engine to extremes that may not be representative of your customer needs, so will the data sets. Different data set characteristics, and the frequency at which end users access that data can make the code perform in very different ways. For example, there are several places within the SharePoint stack, which cache data to save on back end round trips or SQL load. As the number of distinct items requested grows larger, so does the size of the cache. As the cache gets larger it becomes necessary to trim the cache, thus slowing down processing, but also increasing round trips to the back end to re-populated items removed from the cache. Further, the larger the set of popular items in the system, the more likely it will be that there is a cache miss. This means that in order to maintain realistic cache hit, miss and maintenance patterns, the test should be designed to imitate both relative popularity and total quantity of different resources requested. When the data sets and access patterns in tests push the system to extremes beyond what the customer is expected to require the resulting defects are difficult to prioritize. The fix may optimize for the narrow case at the risk of regressions in the more common case.

This motivates the test team to look for real-world production system data that has a corresponding real-world traffic pattern to match. This is difficult, as the production data sets are often larger than the test

team resources can store. Another problem is that production data usually falls under privacy, legal and intellectual property restrictions. Such restrictions may even apply within the same company. For example, the personal site data for employees in a company may not be available outside the IT facility where it is deployed.

1.2.4 INVESTIGATION IN PRODUCTION IS EXPENSIVE, SLOW AND TOO FAR DOWN THE PIPE

The SharePoint team uses internal production systems for long periods to discover issues in the product. While this technique yields a large volume of high value flaw discovery it is an expensive means to find bugs. Downtime on a production system means work stoppage for everybody using the system. Production systems run under complicated conditions and the patterns of usage are unpredictable. Many debugging and diagnostic procedures are unavailable. The code running on the production system is also usually multiple weeks and eventually months behind the code that is in development. All of this motivates a desire to move bug discovery upstream in laboratory tests that simulate the production environment.

1.3 WHAT WE PLANNED TO ACHIEVE

Based on all of these issues, we set out the following goals:

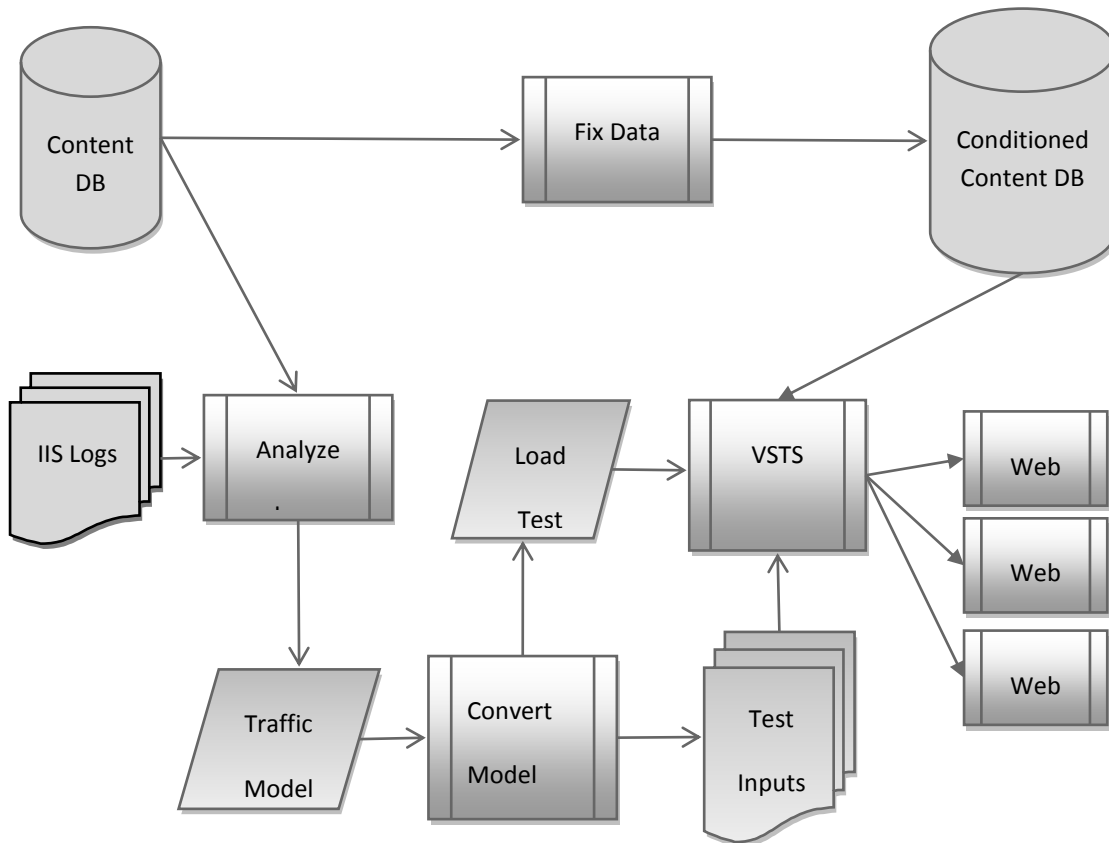
1. Run tests that predict the performance and reliability flaws that manifest on our production systems before deployment
2. Find usage patterns from the real-world samples that manifest bugs that are otherwise hard to discover
3. Simulate real-world traffic patterns taken from existing customer and internal systems to help calibrate our awareness of the importance of different bug fixes, performance goals and objectives
4. Create a regression suite that can be used for non-production problem investigation and code fix validation before checking source code into the project
5. Create a test lab environment that we can use for inventing test methodologies for investigation and diagnosis
6. Re-use the solution to help customers with their own capacity planning and performance investigation activities

2 SYSTEM DESCRIPTION

We built a load simulation tool that was designed to satisfy these needs. This tool was first used in SharePoint 2010. It is designed to sample existing traffic data (taken from Microsoft Internet Information Server logs on a customer machine) and content (taken from existing SharePoint database) and generate the data files necessary to tell a load test what to do. It uses Microsoft Visual Studio Test System (VSTS) to generate, load, and execute the tests. By basing the load component on a shipping product, we achieved portability of the test to any environment.

2.1 TOOL ARCHITECTURE OVERVIEW

There are several parts to the system:



1. Analyze the IIS logs to establish request types and traffic patterns (Traffic Model)
2. Build a model that describes the distribution of request types and resources
3. Convert the model into the input data for the load test
4. Copy the Content DB to the test system
5. Condition the content to make it test ready ("Fix Data" in the diagram)
6. A set of web tests capable of executing different types of operations against the system under test

2.2 ANALYZE THE IIS LOGS TO ESTABLISH REQUEST TYPES AND TRAFFIC PATTERNS

2.2.1 REQUEST CLASSIFICATION

The Traffic Analysis component processes the IIS logs a line at a time and looks for information in the request URL, request type and User-Agent to determine what type of request the user executed. This is important because not all requests can be accurately executed as a generic GET or POST action against the same resource. Specialized test code is required to simulate accurately the scenario, and identifying the type of request is the step that establishes which test code to use to process the request.

The traffic analysis component used a set of pattern matching rules to narrow down the request type. Here is an example of some of those rules:

Test Name	Output File	Object Type	IIS Criteria
ECM_GetStaticFileInDocLib	file.csv	File	csMethod = 'GET' and csUriStem not like '%/personal/%' and csUriStem not like '%/_layouts/%' and substring(csUriStem, len(csUriStem)-charindex('.', reverse(csUriStem)) + 2, charindex('.', reverse(csUriStem))) in ('jpg', 'gif', 'css', 'js', 'pgn', 'axd', 'bmp', 'ico')
WSS_ViewListRSSFeed	list.csv	ListFeed	csMethod = 'GET' and csUriStem like '%/listfeed.aspx'
MOSS_MySite_ViewProfilePage_ByEveryone	accountname.csv	MySitePublic	csMethod = 'GET' and csUriStem like '%/Person.aspx' and csUriQuery like 'accountname%'
MOSS_MySite_SharedDocLib_ViewAllItemsInFolder	doclib.csv	FolderView	csMethod = 'GET' and csUserAgent like 'Mozilla%' and csUriStem like '%/Shared+Documents/forms/allitems.aspx' and csUriQuery like 'RootFolder=%'

In the above example, "IIS Criteria" is a query used to match content found on the HTTP request entry in the server's log file. If the query matches, then the request is assigned to the web test defined in the "Test Name" column. The "Object Type" column indicates the type of action or resource the end user was accessing in the request. The "Output File" column indicates which test data file to store test parameters in for this request.

2.2.2 REQUEST AND USER DISTRIBUTION

The analysis tool also tracks which users and which resources are accessed how often on the production system. This step has to happen after request classification because different parts of the request data may mean different things to different types of tests. For example, some tests may simply require the full URL to determine what to do. Other tests may ignore parts of the URL and look at only a specific part of the path or just the filename. Other tests may require extracting part of the QueryString (the part of a URL request following the question mark) to further isolate specific operations or data locations.

2.2.3 USER PERMISSION ANALYSIS

Another part of the analysis is to traverse the content database and determine which users have what permissions levels inside the content. SharePoint allows per user and per group permissions at the entire site, specific site, specific library and in some cases file level. Operational costs vary a great deal per permission level, so modeling the permissions accurately is necessary to avoid creating unrealistic load patterns.

2.3 TRANSLATE THE MODEL INTO TEST DATA

The output of the analysis phase is a file expressing a model of the percentage distribution of the different operations in the IIS log. The percentage of users and resources accessed are likewise recorded in the model on a per operation basis. The following is an example:

testname	Distribution
Ecm_GetStaticResourcesInDocLib	16.148
Ecm_ViewSiteHomepage	3.621
Wss_ViewSiteHomepage	5.775
Wss_ViewCustomDocLibView	0.921
Wss_ViewListAllItemsPage	0.480

Wss_ViewDocLibAllItemsPage	1.042
Wss_ViewAllSiteContentPage	0.083
Wss_ViewListRSSFeed	1.214
Wss_ViewStandardRSSFeed	0.081
Wss_Download_SaveTargetAs	0.888
Wss_Download_OpenInReadOnlyMode	48.400
Wss_Download_OpenInEditMode	21.347

In the above example, “testname” refers to the name of a web test in Visual Studio (described later in this document), and “distribution” refers to the percentage of time that test should be given during the run. .

The next step is to convert this model into a format that can be consumed by the web tests. There are two parts of this:

1. A Visual Studio Load Test file
Visual Studio load test files describe the behaviors of a test run; which tests to execute, what percentage weighting to give, how to model the load pattern on the test harness, etc. The conversion from model to load test is straightforward, the frequency of the operations identified in the model are mapped directly to the percentage weighting of the tests that correspond to those operations.
2. Test data sources
Visual Studio allows binding between a test and a data source to describe inputs to the test. In this case, the inputs map to the resources and users associated with the operation during the analysis phase. The data sources are constructed such that more popular resources and users that are more active are used in the test run than less popular resources or active users in proportion to what was observed in the real-world sample. In this instance, the data sources were simple text files in CSV (Comma Separated Values) format.

At this point, Visual Studio has enough information about the test model to execute the tests, users, and resource access patterns in the same percentage as was observed in the production system. All that is required from this point are an available site prepared with a copy of the data and a set of web tests able to execute the operations.

2.4 WEB TESTS

Visual Studio Test System uses two types of objects to do a test run. The outer most object is the load test, which is really just a container for mixing test code together in a run different test inputs, percentages and run time conditions. The inner most object is a web test, which actually executes the operation being tested. There are simple web tests, which just play back URL requests and allow test development without any coding, and there are coded web tests, which allow you to introduce complex test behaviors via compiled code that go beyond the capabilities of the simple web tests. The web tests are really where the tests happen. The load test is just a container to coordinate different web tests in a mix.

The web tests are actually an independent part of the system. The SharePoint team was developing them already in order to do isolated load tests against single components and operations, as well as simple mixing. Some examples of web tests include; uploading documents, rendering a spreadsheet online, initiating a workflow on an item in a list, filling out a custom form and submitting it to a list or simulating synching of RSS ([web feeds](#) that publish frequently updated works) feeds to a client application. Several dozen web tests were developed that represented approximate 99.95% of the request classifications that we had seen from traffic in our internal 2007 deployments.

The important part of this system was to author the web tests in such a way that their behavior was driven by input files built from the traffic model. This allowed the tests to adapt to the traffic model rather than impose their own patterns independent of the model.

While not directly part of the real-world modeling system, the bulk of the test development time was spent building and adjusting the web tests. As noted later in this document, feature and operational coverage are a significant part of the value of the pattern. The value of the test run diminishes substantially for every operation, or permutation on operational behavior not represented with a test. Making this test development even more difficult is the fact that SharePoint is built by approximately twenty different teams within the Office organization. Coordinating the development, delivery, analysis and maintenance of these components took a great deal of effort.

2.5 CONTENT AND DATABASE COPY AND PREPARATION

The original content database copied to the test system must be modified before test execution. A copy of the actual data is used so that when the web tests execute they request resources that actually exist, but the data in its original form is not ready for test.

User permissions must be changed on the database before the system can be tested. The users from the production system are real users that live in an actual network. Their user identities are mapped to security properties that determine their permission within the system. However, the password and credentials of those users are not available to the test system. This means that the real users must be replaced with artificial users coming from an Active Directory inside a domain that is under control of the test system.

This is where the permission modeling comes in. The test accounts are mapped to real users from the model, and then the test accounts are mapped to groups. Those groups are then given permissions inside the copy of the content database that mimics the distribution of permissions of the real users in the original production system. This condition allows the web test to log into the test system with the artificial test user credentials and have the same permission levels of the users from the production system.

2.6 PERFORMANCE AND RELIABILITY MONITORING

We included an in-house performance and reliability monitoring solution during the runs in addition to Visual Studio's own data collection. This monitoring solution was the same one we were using on our production system, allowing us to gather the same statistics during production and lab runs and compare results.

The monitoring tool executes a "ping" in the form of an HTTP request against a known set of URL's on the system under test. The result of this ping was used to measure time to last byte in the request, errors, and timeouts. These were aggregated to collect data on page latency percentages (time to last byte at 25%, 75% and 95% of the sample), availability (percentage of time the service was up versus down) and failure rate (percentage of requests returning an error or timing out).

3 APPLICATION

We used this load simulation system, or parts of it multiple times during the SharePoint 2010 development cycle. In addition to modeling real production systems, it also afforded us the opportunity to re-sample the traffic pattern and update the model to accommodate changing usage patterns as new features are brought into production.

Below are some examples of real-world deployments we modeled.

3.1 OFFICE DIVISION PORTAL SIMULATED LOAD TEST

The Microsoft Office team uses a SharePoint portal for its own divisional point of access to a variety of information and tools. Office has approximately 7,000 users in the organization, and the site is used to

communicate information about Office to the rest of the company, coordinate project development and communication about new versions of Office to the Office team, coordinate communication and collaboration with partner teams within Microsoft and serves as a place for teams within Office to coordinate their individual features and projects. The site has archives of data and documents regarding previous projects, which are used for sustained engineering and support. It also has a single library that is used to house all of the feature requirements, design documents, tests specifications, and general development collateral for an entire release of Office. The site receives about 10,000 unique visitors a week and about 7500 unique visitors a day. In addition to servicing collaboration needs of the division, the site also has personalized site and user profile features that would normally be shared across an enterprise – this last aspect of the site is done to give the Office team an opportunity to put the software into production and experiment with feature behavior before the entire company adopts the features.

In releases previous to this Office 2010, deploying this pre-production system took considerable time. A substantial part of this effort was addressing performance and stability issues. To mitigate this in the 2010 release, one of the goals of the performance testing effort for Office 2010 was to test the Office portal site ahead of the production deployment, allowing an opportunity to identify major performance issues early and fix before going into production.

This was the first system that was sampled, modeled, and simulated in the load test system. Most of the necessary web tests for the remaining samples were developed to satisfy the Office portal site.

3.2 MICROSOFT IT'S HOSTED COLLABORATION PORTAL SIMULATION, AND DEALING WITH PRIVACY ISSUES

The next major sites we sampled were two team collaboration sites hosted by Microsoft's own IT group. These two portals are a location where any team within Microsoft can self-deploy their own site. The sites receive approximate 80,000 unique visitors per day.

The usage patterns between the IT hosted solution and the Office divisional portal are similar, but differ in important ways:

1. The Office site represents collaboration of a single division, which means that the patterns of usage vary as the Office team moves through its project schedule. This means that there are times of the year where document authoring was far more common, and then long spans where very little document authoring happened at all.
2. The IT hosted site represents the collaboration of hundreds of teams, all at different stages in their project cycles and with different classes of business needs. This spreads the usage pattern more evenly across time, meaning you do not see as much seasonal variation in usage.
3. The Office team is worldwide, but predominantly resides in the Redmond, WA campus. This means that while there are usage humps during business hours in Asia and Europe, they are not as pronounced as the ones in Washington.
4. The IT hosted solution represents a far larger distribution of employees outside the Redmond, WA area, which means a larger usage hump as the business hours move across the globe.
5. The Office site user behavior has a higher usage rate (Office users on average access the Office site more often than users of the IT solution did), but the volume of users accessing the IT site on a daily basis push the IT system to about twice the throughput rate of the Office team solution (Office site at 8000 users with 155 Requests Per Second (RPS) peak, IT hosted site with 80,000 users at 304 peak RPS).
6. No other team on the IT hosted solution has created a single document library as large as the Office specification library. The activity on the IT hosted solution tends toward smaller lists and libraries.
7. The data on the IT hosted solution falls under more sensitive privacy and legal restrictions.

For the most part, the IT hosted solution represented an opportunity to observe traffic patterns that we felt represented a more typical case that tracked more closely to what many customers would experience. It

also represented a chance to run our software on a production system with more rigorous business demands and hence less tolerance for problems and failure than with the Office product group.

From an engineering perspective, however, the site represented a more interesting challenge. The data set overall on the IT hosted solution was much larger than we had room to host in the test laboratory. Further, the legal and privacy restrictions prevented us from copying the entire content database out of the production facility and into our test laboratory. We were only allowed to copy an approved subset of the data to our test laboratory and work with it from there.

This forced us to modify the load simulation process slightly. We still wanted to represent the volume and percentage mix of users and resources based on a real system, but we could not use all the content data to do it. Therefore, we added an incremental step. We took the model described by a sample from the full system, and then mapped the resources described in that model to just use the subset of the data we were able to copy to our lab. This allowed us to describe the traffic pattern we wanted and satisfy the legal, privacy and size limitations of using the production data.

This created a problem. The smaller data set meant we were not exactly modeling the scale characteristics of the system. However, we did have larger scale testing results on artificial load and data sets, just on single variables at a time. For example, we had tests that measured how the number of files in a library affected performance, and another that measured how the number of sites inside a website-affected performance, and likewise for other variables, but not how all those variables affected each other in combination. We used these results on independent variables and extrapolated back to our real world test run to assess how we expected the real world performance to behave as it went to larger scale levels. We also relied on aspects of the system design that we knew could scale independently. For example, the total number of sites in a SharePoint system can grow infinitely so long as you continue adding databases and database servers to handle the data set, so we knew that as long as we established the scale potential of a single database server we had enough information to infer the rest of the scale requirements without building an exact replica of the real world deployment. These extrapolations were obviously compromises, but were ones we believed came with manageable risks.

3.3 MICROSOFT'S IT HOSTED PERSONAL SITE SIMULATION

SharePoint has a feature that allows hosting of personal sites for all employees within an Enterprise. The personal site has a profile page for every person with data about them regarding office location, location in the organization hierarchy, expertise or any other piece of information the company chooses to track. Likewise, each user can have a site for keeping personal documents, documents they share with others, lists to track activity and collaborate with others.

Microsoft IT hosts a personal site portal for all employees with the company. There are approximately 150k employees hosted on this portal. The site receives approximate 71,000 unique users per day generating an average of 93 RPS. By contrast, the MS IT hosted collaboration portal, at 80,000 unique users and 304 RPS generates approximately 3 times the load per user.

The traffic on the IT site is dominated by client synchronization requests for RSS feeds, and in the Office 2010 release activity feed and social network updates from Outlook. Part of the reason for the high volume of RSS feeds is because Microsoft IT has configured a default synchronization feed to every user's client to their shared document library.

The personal site data has a similar size and security policy problem as the hosted collaboration solution, with the end user privacy issues being even more restricted. The test-engineering problem was very similar, we had to map real users to artificial users in order to obscure identity, and we were only permitted a very specific subset of the user data.

An even more difficult problem arose regarding personal site visits. When a user visits a portal site, their identity is detected and they are shown a profile page. This page renders differently when a user is viewing their own page versus when they are viewing someone else's. However, since we were using

artificial test accounts to run the tests we were not able to visit a person's own profile page. To solve this problem, we gave the artificial accounts the same permission as the site owner to the personal site of each real user.

3.4 CAPACITY PLANNING TOOLS AND DOCUMENTATION

Customers that purchase and deploy SharePoint need to purchase hardware and plan their configuration. They require a set of documentation and tools that help them determine what hardware will meet their needs, and how to properly configure and tune that hardware once deployed. Such a set of tools were created and published with SharePoint. The test tools described in this document were a critical component in preparation of that toolkit and documentation.

3.4.1 CAPACITY PLANNING REPORTS

The capacity planning reports and documentation for SharePoint 2010 used the load test solution described in this document. These reports are available on the web at the following locations:

All capacity planning case studies can be found here:

<http://technet.microsoft.com/en-us/library/cc261716.aspx>

Site From This Document	Report name on website
Office Product Group Portal	Departmental Collaboration
Microsoft IT Hosted Collaboration Portal	Intranet Collaboration
Microsoft IT Hosted Personal Site Portal	Social

Further, load pattern and resource usage analysis reports of the actual production system are included in the same website. Basing our capacity planning processes on loads modeled from real production systems allowed us to show impacts of scalability and load changes and then allow customers to compare them to the real system. This gives the customer some visibility into the difference between the test run data (which is somewhat artificial) and the real experience, thus we hope making it easier to calibrate the test data and make their own planning easier.

3.4.2 LOAD TEST KIT & EXTERNAL CUSTOMER EXPERIENCE

Before release, the testing tool was packaged as a customer consumable load testing kit. The idea being that the customer could base hardware purchases and configuration choices on real traffic patterns from their existing deployments rather than making blind guesses. We had an opportunity to work with one of the customers on the early adoption program with this test kit.

An interesting example of an issue that came up during this customer test is one similar to something we experienced internally. The customer had 3 terabytes of data in their production system, but was only able to use about 500 gigabytes in the test run due to cost constraints. The log file that the test kit analyzed, however, contained requests for the entire production data set. The solution to this problem was similar to our own; requests for missing data were mapped to test data that existed in the laboratory.

Other challenges came up that centered mostly on education about the test framework, familiarity with performance planning in general and extending the test suite to cover additional behaviors. Load testing and capacity planning are complex problems, and extending what you have learned to do before release out to your own customers is a difficult task.

In general, though, we found the tool an effective aid to help the customer to plan their deployment and make hardware decisions.

4 DEFECT FIX AND FIND RATES

Simulation Runs

The simulation runs were considered a very successful project, resulting in a high defect find rate. In the table below are the bug numbers found by two of the simulation tests mentioned in this document (the Office team portal and the IT hosted collaboration portal). Also included are defect resolution percentages, which refer to whether the defect was ultimately fixed or not. The resolution legend is as follows:

By Design: The behavior reported is intentional and not a bug

Dupe: This problem has been reported already

Ext.: This behavior is due to a defect in an external component

Fixed: The defects that were fixed

Not Repro: Attempts to reproduce the problem failed

Postponed: The defect will be reconsidered for fixing in a later release

Won't Fix: The defect will not be fixed

There are two rows of bug totals, one for performance related defects, and "Other" which refers to not-performance defects found as a side-effect of the performance testing process such as pages that failed to render, errors found in product logs during execution, failed data set upgrades, or other various functionality related failures.

Simulated Test Run Defect Find and Resolution Rates

	By Design		Dupe		Ext.		Fixed		Not Repro		Postponed		Won't Fix		Total #
	#	%	#	%	#	%	#	%	#	%	#	%	#	%	
Performance	25	8%	84	27%	11	4%	88	28%	43	14%	3	1%	55	18%	309
Other	13	10%	30	23%	5	4%	57	43%	20	15%		0%	8	6%	133
Grand Total	38	9%	114	26%	16	4%	145	33%	63	14%	3	1%	63	14%	442

It is interesting to compare the resolution rates to similar bugs found by different test methodology. The following chart shows resolution of all performance defects, filed against the same teams as in the chart above, during the same time period.

Performance Defect Resolution Rate Overall (from a total set of 19269 reported defects)

	By Design	Duplicate	External	Fixed	Not Repro	Postponed	Won't Fix
%	4.21%	14.39%	1.90%	47.49%	8.31%	3.60%	19.83%

There are some substantial differences in this data. The simulated runs found about 1.6% of the total performance defects (309/19269) during the same time period. This is partly due to the complexity of the simulated runs, but mostly due to staffing differences. The simulated real-world runs were executed and defects filed by 27 (11 per) testers, whereas 1521 (12 per) testers filed the remainder of performance defects, so the bug yield per person was comparable for the same class of defects.

A larger difference can be noted in the resolution rates. For performance bugs overall, the fix rate was 17% higher than on the simulated runs, picking up substantial differences in Dupe (-13%) and Not Repro (-6%). The reasons for these differences are not well understood at this point, but likely deserve further analysis.

5 LIMITATIONS & OPPORTUNITIES FOR FURTHER DEVELOPMENT

While the solution provided much value, there are still opportunities for improvement. Some of those are described here.

5.1 COVERAGE LIMITATIONS

The impact of accurate modeling is very rapidly overwhelmed by the limitation of missing coverage. The system is only capable of testing requests for which there is an existing web test written. For simple requests, this is easily handled by playing back the exact request, but for more complicated requests the missing edge conditions and user scenarios have a large impact on the performance characteristics of the system.

What we have observed is that single actions have a dramatic effect on performance. The software overall may be well within performance goals, but a customer may still experience bad performance when less frequent but very expensive events cause momentarily high latencies or outages. The causes and classifications of these events are varied (bad database queries, synchronous events blocking on serialized resources, memory leaks, etc.) The challenge in their discovery is in creating tests that will reproduce such conditions.

There are a couple of examples in areas where missing coverage contributed to defects that escaped testing and didn't show up until production:

Co-Authoring: SharePoint 2010 supports a file save, open and editing protocol that enables multiple users to work on a document at the same time from different clients on different machines. The Office 2010 client applications utilize this protocol. The protocol is very rich and complex, and correct use of the protocol requires full understanding of the file formats being edited. This complexity made it difficult to deliver as broad a set of web tests as were necessary to identify performance and reliability problems. Several critical flaws escaped runs in the test environment and manifested the production MS IT-hosted collaboration portal before they were discovered by real end users and eliminated by the product team.

Profile Synchronization: The personalization features in SharePoint 2010 allow synchronization of user data from external data stores (Active Directory, LDAP, SQL databases, SAP, etc.) to a SharePoint profile store that is used for page rendering, local business application logic, and other scenarios. This synchronization activity impacts the same resources that are used to deliver user data. The desired test was to run this synchronization in the lab concurrent with simulated for end user traffic on the personalized site portal. Schedule and costing issues delayed development of this specific test, which again meant that several issues were not discovered until Microsoft IT's hosted portal deployed the full system.

Test automation gaps are nothing new or unique, so the point is not to confess that we experienced the same thing. The point is to raise the question of priority of more coverage over more accurate modeling. It seems reasonable to assume that accuracy in load modeling need only get to a "close enough" level of precision. By contrast, establishing "good enough" on coverage requires a great deal more effort, and seems like missing it comes at a higher cost. Perhaps the question merits more investigation, but when it

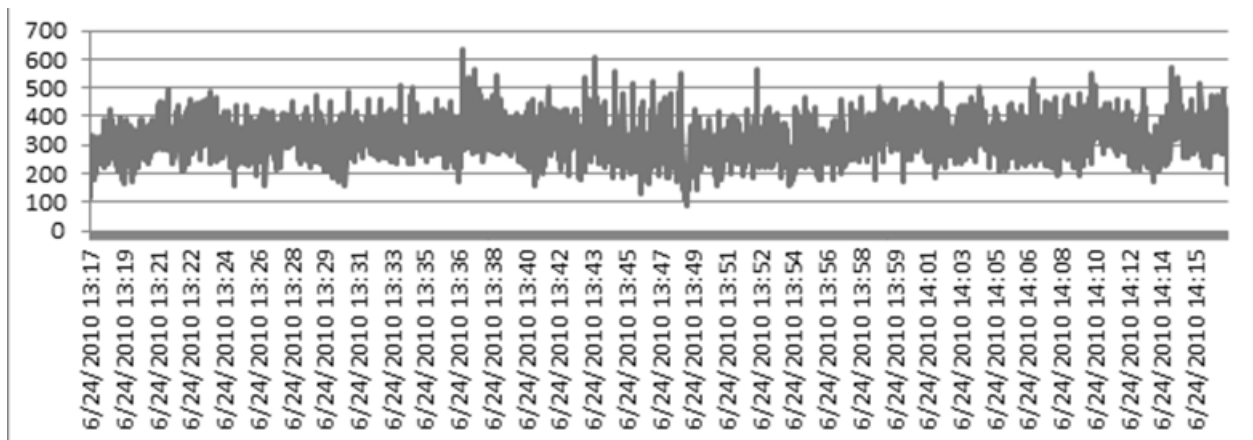
comes to choosing where to spend the money, it seems best to put most of the effort into achieving as much coverage as possible and make trades in the accuracy of the load simulation distributions once they are close enough.

5.2 TRAFFIC PATTERN FLATTENING

The traffic analysis phase of the tool in this document models frequencies as a percentage. If there were 100 requests sampled in an hour, and an operation occurred ten times, then the model will set that operation as 10% of the load.

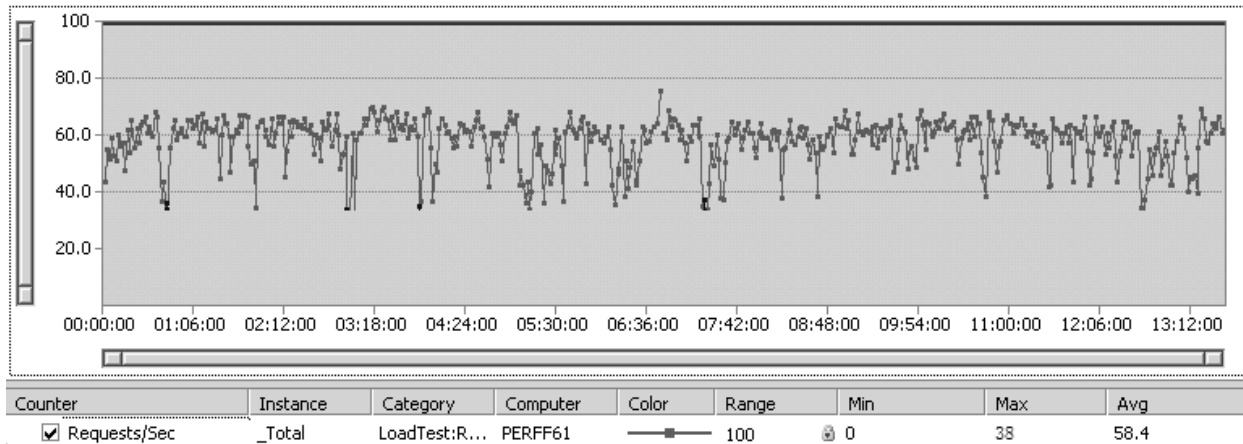
This approach does a pretty good job of modeling typical load and resource usage. If the RPS volume is taken up close to the system maximum it does reasonable job of predicting system resource utilization (memory, CPU, disk IO) for that load.

What it fails to represent are bursts and spikes in requests that happen over very short intervals. A typical graph of RPS for a production system looks like the following (taken over a one hour basis)



As can be observed from the graph above RPS vary widely over time, with as much as a six times difference between the high and low points. Average values per hour, per minute and per second have increasingly larger variances.

By contrast, the RPS pattern from the simulation is relatively flat and constant



The line in the above graph center stays much closer to the average of 58.4 requests per second. The maximum value of 58.4 requests per second is only one and a half times higher than the minimum value of 38 requests per second.

The reason for this is the following:

1. The load test examines traffic one request at a time and not in groups, hence requests don't cluster together in the simulation the way they do in the real world.
2. The load test model flattens all variations in the pattern to a percentage, which then spreads request rate evenly over time.

The concern is that the simulations are missing spikes in resource utilization, so while the general resource consumption is close to production need, the momentary spikes that cause problems are not represented. What is not understood yet is the significance in this gap in the simulation.

It could be that modeling these variations in usage as clusters and spikes may or may not yield any substantial discovery of performance defects via the test simulation. That said, further investigation of ways to represent spikes more accurately in the model might turn out to be worth the investment.

5.3 IMPROVED COST OF EXECUTION

Something not mentioned in this document so far has been the cost of creating the test environment. Every attempt was made to create a deployment that was similar (if not identical) to the configuration used in the production systems. Even with simplifications of the configuration, the resulting test environment was very expensive to create, deploy new code to, apply settings, load with data, and generally get ready for execution.

In the previous release, test runs on environments of similar complexity took weeks to build and prepare. A great deal of time was lost discovering and attempting to mitigate product bugs that prevented deployment or that blocked testing. For example, there were numerous times during the schedule when data set upgrade defects in the product would manifest on the data sets used in the performance tests, causing the upgrade of the data to abort, hence delaying the entire test until a fix was delivered for the upgrade failure. However, even with those issues resolved, a substantial part of the cost was in the complexity of setting up and configuring the test environment.

Deployment went from taking weeks down to approximately four hours. We achieved this by automating the process of copying data, installing the product, and setting up the basic configuration. The difference in time was critical, because it permitted us to begin a simulated test run within one day of a build either from the main development pipeline or from private releases from a single developer. This allowed us eventually to use the simulated runs more often for regression testing, investigating issues, bug reproduction, and eventually preparing capacity planning documentation for customer consumption.

Regarding priorities, there is a temptation to invest only in the test code and leave steps like installation, configuration, and environment preparation to test personnel or lab staff. Our experience was the delays and costs made such tests prohibitive and of marginal value. The recommendation, then, is to invest substantially in efforts to automate quick and rapid deployment of test ready environments. Again, it is likely this will have more return on value than pursuing high accuracy in the test simulation.

5.4 LARGE RETURN FROM MONITORING INVESTMENTS

The SharePoint team invested in a number of monitoring solutions, some built into the product, and others built as external tools. These monitoring systems gave us data on availability, request failure rates, percentile spreads on page latency, system crashes, and a variety of other metrics on system reliability and performance. We also built a number of tools that permitted deep, rapid investigation into source and root cause of failures.

We deployed these monitoring solutions on all of our internal production systems. We used the same monitoring solutions on our test runs. This allowed us to compare our metrics from a test run to production systems. It also gave us the same toolset for investigating root cause in lab runs as in production, which made reproduction and regression testing easier.

Even more important, though, was the efficiency at finding and filing bug failures. Before tool creation, filing a single performance bug would take days of mining data, looking at log entries and analyzing results. This often resulted in a bug that development was unable to diagnose. Improved monitoring and analysis tools dropped the costs from days to minutes and increased the bug discovery quantity a hundred fold.

This set of monitoring tools was critical to interpreting the results of these simulated load tests. The complexity of the run is close to the complexity of production systems, and the volume of requests to the system made manual investigation completely untenable. Like some of the other investments mentioned in this document, the monitoring and analysis component investment pays off more rapidly than accuracy in modeling.

6 CONCLUSION

Simulating real-world patterns is a challenging, but necessary part of performance and reliability testing for a server product. Simple playback and record technologies don't adequately capture the full complexity of the scenario and often are unable to simulate dynamic run-time features. This necessitates creation of specialized load tests that are aware of the product itself and which can handle the simulation appropriately. What we discovered is that while building such tools based on real world traffic samplings is difficult, it is definitely possible, and the return on investment is high. It is possible to achieve higher defect find rates and better confidence in triage priorities of the defects discovered. Further, such tools can be re-purposed effectively to accommodate different customer usage patterns and different engineering needs that go beyond just getting performance measurements.

We also discovered that this high return on investment could be achieved without establishing a perfect simulation model. There is a substantial difference between "good enough" to be worth the effort and achieving absolute accuracy. We discovered that we had high return on investment from supporting aspects of the solution, such as creating test environments more quickly and easily than doing so manually.

Several interesting questions remain after this exercise. We still do not know how much return on investment we will get on defect discovery costs and rates through more accurate simulations than we already have. We also have not thoroughly analyzed the qualitative difference between bugs found via real-world simulation versus those found with artificial data set and load testing, and don't know yet what to make of the large difference in number of defects found via the two test approaches.

We know now that making an investment in real-world simulation testing was well worth the time, and that we will continue with it as a key part of our test strategy in the future.