International Journal of Computer Applications Technology and Research
Volume 6–Issue 1, 13-18, 2017, ISSN:-2319–8656

Software Performance Workload Modelling

Vijay Datla

Abstract: The debate between performance engineers and business stakeholders over non-functional requirements is probably as old as the performance discipline itself. 'What set of transactions is enough to represent my system?', 'Why do we not load test every transaction?' , 'Our volumes are much higher than what the targets show' are some of the common questions that need to be answered. From a technical perspective, benefits from load testing every transaction are not enough to justify the effort involved in the exercise. However, for a business, even a small risk of one untested low volume transaction affecting the others or bringing down the entire system is high enough to raise a flag. This paper is an attempt to balance these concerns by discussing how to create workload models that are closer representations of the real world enterprise applications. It answers common requirement gathering questions like where to look for information, on what basis to include and exclude use cases from workloads and how to derive a complete and convincing workload model. This paper highlights the risks associated with selective modelling and the possible mitigations. It also brings to the table tips and tricks of the trade, some lessons learnt the hard way.

Keywords: Performance, Modelling, Vijay Datla, Vijay, Datla

1. REQUIREMENT ANALYSIS:

Just like any Software Development Lifecycle (SDLC), a Performance lifecycle also begins with Requirements Analysis with the difference that the requirements are purely non-functional in nature. Non-functional requirement is a requirement that specifies the criteria that can be used to judge the operation of a system rather than a functional behavior. There are several kinds of non-functional requirements like Security, Maintainability, Usability and so on but the specifics that we are interested in are Performance, Scalability and to a certain extent Availability. Requirements gathering forms the foundation for all future performance engineering activities on a project. Mistakes made in understanding the business requirements translate into setting of wrong goals and takes all the performance efforts into the wrong direction. Requirements gathering is therefore the key to a successful Performance Engineering project. But even before getting into requirements, it is important to understand the objectives. It is a common misconception that performance can only be done to measure the response time of the system. In literal terms, measuring performance of a system is purely Performance Testing which is part of a larger discipline called Performance Engineering. Performance testing is a means; an enabler in achieving the Performance engineering objectives. So what are these objectives?

•Measuring and improving Performance of an application
•Meeting the non-functional requirement targets
•Improving user experience
•Benchmarking the application and hardware
•Validating Hardware Sizing Once the objectives are clear, the next step is to define the scope at a high level, meaning which modules or what part of the solution will need to be tested as part of the performance exercise. To go deeper into the objectives and scope of performance, it is essential to have a thorough understanding of the system. This understanding can come not just by studying the application but also by studying the business.

1.1. Asking the right questions:

- Customer base
- Growth rate
- Concurrency
- Volume centircs vs user centrics
- Most common transactions
- Response time requirements
- User arrival pattern

Gathering requirements for performance testing is the most challenging task given that there is no one place with consolidated information and most sources are external. Readily available non-functional performance requirements and statistics is a rare occurrence. However, it not the lack of availability that adds to the challenge, it is the process of gathering and consolidating data from various sources that's a cumbersome task. More than getting the right answers, it is about asking the right questions. Since the process involves dealing with business, its important to frame questions more comprehendible to a business mind. Instead of asking what is the concurrency or throughput target, try asking what is the customer base of the business? How many of these customers will be accessing the system at any given time? The following should give an idea:
• What is the expected business growth rate?
• Is the system volume centric or user centric?
• What response time is the system required to serve in case of web based OLTP transactions?

• What are the most common use cases? or transactions that happen on the system most frequently?
• Do all users arrive into the system over a small window or are they spread across the day?
• What are the peak periods of access to the system?
• Are there periodic tasks that the system is designed to accomplish?
E.g.
• End of Month/Quarter reports?
• Close of business?
• Seasonal sales?
• Year End closing? And so on

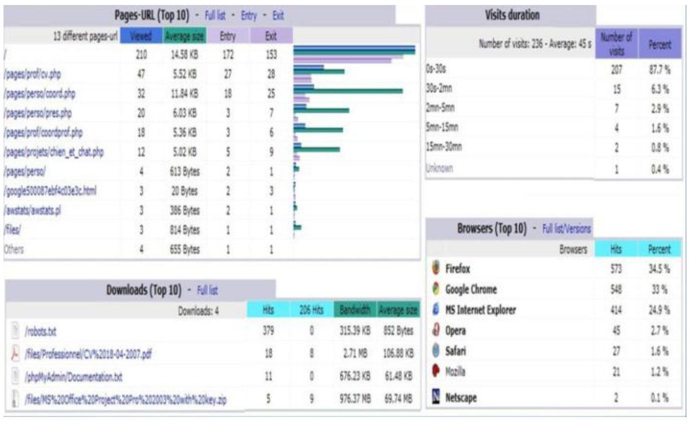
1.2. Picking the right resources:

Common sources like Business Analytics, RFP, Business volume reports, Audit reports, Inputs from legacy system, Capacity sizing document, Webserver access logs, Data ware house, google analytics.On enterprise level projects there can be several sources of information when gathering the non-functional requirements.

•Business Analysts (BAs)
•BAs are always the first source of information for non-functional requirements. They may or may not have all the information required, but they will be able to make the connection to the right business contacts.
•RFPs
•RFPs usually contain a non-functional requirements section. The requirements specific to Performance may be few and non-elaborated but will still contain response times, customer base, transaction volumes etc.
•Business Reports
•There are several reports that the business maintains like Volume reports, Accounting, auditing reports that can provide insight into business statistics
•Legacy Systems
•In case of legacy modernization projects, there already is a system, maybe a mainframe that is still serving the business. Running simple select queries on this system can help in studying the real world transaction volumes and load patterns
•Hardware Sizing Documents
•In the initial stages of SDLC, enterprise projects go through the process of determining the hardware required to support the solution. This sizing is based on the throughput that the system is expected to achieve. So either on a high level or in detail, some study is already done at this stage that can often be used as opposed to reinventing the wheel.
•Google Analytics
•For enterprise applications with already existing websites, Google Analytics is a web-analytics solution that provides detailed insights into the website traffic. It reports traffic patterns, sources of incoming load, navigation patterns, detailed load patterns over a period of time and much more.
•Data Warehouse
•Most enterprise projects maintain data warehouses for storing archived information that can be accessed to obtain non-functional details

•Domain Research
•In most cases there is existing research in the market that has been done on various kind of applications catering to several domains. If there is absolutely no information available in house then these researches can be a good place to start from.
•Log Parsers
•In case of implementations with an existing system in place, server access logs are excellent sources of real time information. There are several tools in the market that parse access logs into comprehendible, meaningful information. There are several log parsing tools in the market that can produce meaningful data from Web Server logs. AWStats is one such open source log parsing tool that is being used here as an example. This parser extracts data from a web server access log and converts it into meaningful server statistics. It is much like a web analytics tool, only that it works offline. It produces graphs that provide insight into load patterns in terms of user visits, page visits bandwidth etc.
The tool lists the most commonly accessed pages which helps in determining the high volume transactions. It also indicates the browser most commonly used to access the application website. With the advancement of browsers features and variety in the market, this information is useful in deciding what browser to use when simulating load on the application.
The most important use of the tool is in studying the user arrival and load pattern. The hourly graphs outline the user arrival pattern and the
weekly, monthly and yearly graphs help in determining the peak periods.

The below charts show the website usage patterns in terms of top accessed URLs, top downloads, average user visit durations and distribution of browsers for incoming requests.



Pages-URL (Top 10) - Full list - Entry - Exit

13 different pages-url	Viewed	Average size	Entry	Exit
/	210	14.58 KB	172	153
/pages/prof/cv.php	47	5.52 KB	27	28
/pages/perso/coord.php	32	11.84 KB	18	25
/pages/perso/pres.php	20	6.03 KB	3	7
/pages/prof/coordprof.php	18	5.36 KB	3	6
/pages/projets/chien_et_chat.php	12	5.02 KB	5	9
/pages/perso/	4	613 Bytes	2	1
/google500087ebf4c03e3c.html	3	20 Bytes	2	3
/awstats/awstats.pl	3	386 Bytes	2	1
/files/	3	814 Bytes	1	1
Others	4	655 Bytes	1	1

Visits duration

Number of visits: 236 - Average: 45 s	Number of visits	Percent
0s-30s	207	87.7 %
30s-2mn	15	6.3 %
2mn-5mn	7	2.9 %
5mn-15mn	4	1.6 %
15mn-30mn	2	0.8 %
Unknown	1	0.4 %

Downloads (Top 10) - Full list

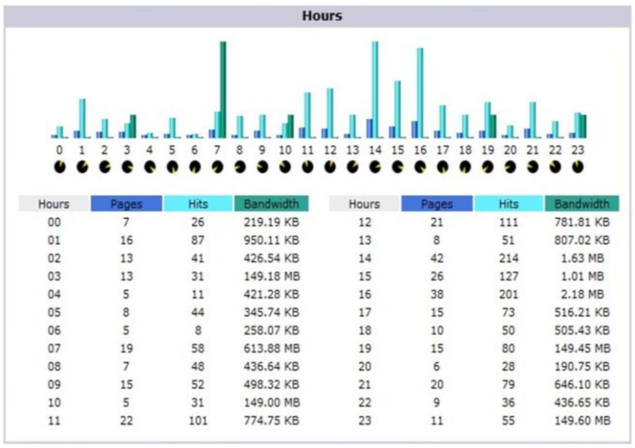
Downloads: 4	Hits	206 Hits	Bandwidth	Average size
/robots.txt	379	0	315.39 KB	852 Bytes
/files/Professionnel/CV%2018-04-2007.pdf	18	8	2.71 MB	106.88 KB
/phpMyAdmin/Documentation.txt	11	0	676.23 KB	61.48 KB
/files/MS%20Office%20Project%20Pro%202003%20with%20key.zip	5	9	976.37 MB	69.74 MB

Browsers (Top 10) - Full list/Versions

Browsers	Hits	Percent
Firefox	573	34.5 %
Google Chrome	548	33 %
MS Internet Explorer	414	24.9 %
Opera	45	2.7 %
Safari	27	1.6 %
Mozilla	21	1.2 %
Netscape	2	0.1 %

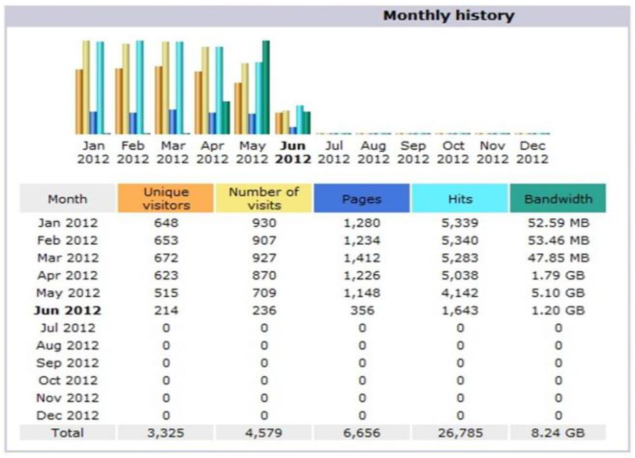
The below graph shows the hourly distribution of load.
This kind of information helps determine the peak hours of the day and the % increase in load during the peak

hour. Having access to this information also helps in deciding off peak windows for scheduling batch and cron jobs during the day.



Hours	Pages	Hits	Bandwidth	Hours	Pages	Hits	Bandwidth
00	7	26	219.19 KB	12	21	111	781.81 KB
01	16	87	950.11 KB	13	8	51	807.02 KB
02	13	41	426.54 KB	14	42	214	1.63 MB
03	13	31	149.18 MB	15	26	127	1.01 MB
04	5	11	421.28 KB	16	38	201	2.18 MB
05	8	44	345.74 KB	17	15	73	516.21 KB
06	5	8	258.07 KB	18	10	50	505.43 KB
07	19	58	613.88 MB	19	15	80	149.45 MB
08	7	48	436.64 KB	20	6	28	190.75 KB
09	15	52	498.32 KB	21	20	79	646.10 KB
10	5	31	149.00 MB	22	9	36	436.65 KB
11	22	101	774.75 KB	23	11	55	149.60 MB

The below graph shows the distribution of load over a year. This information helps in understanding seasonal workloads if any experienced by the business and in turn the application.



Month	Unique visitors	Number of visits	Pages	Hits	Bandwidth
Jan 2012	648	930	1,280	5,339	52.59 MB
Feb 2012	653	907	1,234	5,340	53.46 MB
Mar 2012	672	927	1,412	5,283	47.85 MB
Apr 2012	623	870	1,226	5,038	1.79 GB
May 2012	515	709	1,148	4,142	5.10 GB
Jun 2012	214	236	356	1,643	1.20 GB
Jul 2012	0	0	0	0	0
Aug 2012	0	0	0	0	0
Sep 2012	0	0	0	0	0
Oct 2012	0	0	0	0	0
Nov 2012	0	0	0	0	0
Dec 2012	0	0	0	0	0
Total	3,325	4,579	6,656	26,785	8.24 GB

2. DEFINING SCOPE:

Consider high volume, Complex design, Business Impact, Resource Intensive, Seasonal peaks. With this gathered Information define the categorization and target for combined use case volumes in each category and test the high volume use case in each category.

1.3. Categorization

•One other premise that can go a long way in maximizing the code coverage of performance testing efforts and de-risking the system is categorization.

•Several enterprise transactions can be classified as variations or flavors of one base transaction. Even though there will be slight variation in input parameters, the backend tables and the data access objects will be the same. For instance, a customer updating his phone number vs. updating his address in the profile. Even though both transactions start out differently, they essentially perform an UPDATE on the profile table, and one can be termed as a flavor of the other.

•Along similar lines, the transaction could be the same but coming in from different sources. E.g. a request for account creation could come in from the web, from an agency or from customer service agents over the phone. However different the sources, the execution flow in all cases would involve a call to the same WebService and would end in an INSERT in the accounts and related reference tables.

•Once you have identified sets of similar transactions, combine the volumes of each; select the transaction with the highest volume to represent the set; and load test it to the combined volume.

•This approach covers wider grounds while limiting the effort involved in preparing and maintaining test frameworks for each transaction.

Most complex enterprise applications today are heavily data dependent. A simple example of such a transaction would be funds transfer in a bank account. To complete this transaction, there is a pre-requisite of having enough funds in a source account. If we keep executing this transaction over a set of accounts, the data will need to be refreshed either by using a different set of accounts or by changing the available balance on existing accounts.

To make it more complex, there are systems like Service Request Management Systems that are designed around flow of data from one stage to the other. Performance testing such systems becomes a nightmare because one successful execution of tests requires useful data to be created at each stage and the entire cycle repeated for the next run.

This added complexity introduces another factor which is the Return on Investment i.e. whether the effort involved in preparing for and maintaining a test case from one run to the other is worth the benefit from testing it.

In essence, it cannot be just one factor that can sufficiently determine the transaction set but it has to be a combination of all. Whatever the selection process, the choices are influenced by aggressive delivery schedules and there is always a trade-off.

3. CREATING WORKLOAD MODEL:

Factors to be considered are growth rates, transactional distribution, complex transactions.

When defining targets it is important to account for growth rate. Non-functional requirements are usually defined in the initial stages of the project. By the time the solution goes to production business volumes grow considerably. The targets defined for performance testing should be raised by the growth rate factor up to the roll out dates.

For a simple system where most transactions take the same amount of time to complete, the conversion from throughput to concurrency and vice-versa can be generalized to a simple formula:

$T=C/(tt+rt)$

Where T is the throughput (tps) in page views per second

C is the concurrency

tt is the think time between pages in seconds

And rt is the Response time of each page in seconds

However, in case of complex longer transactions the workload model has to be worked out differently.

Let us take an example of a generic Core Banking Application. A core banking solution will comprise of several modules that cater to Teller Banking, Online/Net Banking, Tele Banking, Mobile Banking, Customer Service etc.

All these modules function as different entry points into the system. Despite the different interfaces and web layers, they will all access the same backend services, data objects and database tables. So if we were to define scope and create a workload model for this application, we will have to look at the architecture on a whole by considering requirements of individual modules and how they interact with each other and the external interfaces.

Unlike functional testing, performance testing efforts have to be limited to only a select number of transactions. Before deriving a workload model, we have to first select transactions that form the scope of performance testing within each module.

For simplicity, let us work with three modules of our core banking application- Teller Banking, Online Banking and Phone Banking.

Functionally, there are a total of 22 use cases arising from these modules as listed in the table above. For a business, the ideal risk-free scenario is to performance test all 22 use cases. However, the effort involved in creating a load test framework for 22 Use cases and maintaining it across builds and releases can be a very challenging and time consuming activity. Projects seldom have the resources and the time to support the ask. Moreover, the benefit from load testing every use case is usually not worth the effort involved.

So we need to draw a line at a certain throughput, i.e. define a threshold below which a use case will not be considered for load testing. Use cases highlighted with green in the table above are transactions chosen on account of their high volumes.

Now that we have defined the scope, we will derive a workload using the requirements and data available. In most enterprise applications, the requirements are a combination of volume-centric and user-centric targets, i.e. module level concurrency and business volumes targets for every use case. For instance, in our example of the Banking application, its easy to know how many bank tellers will be using the core banking application, how many customer service agents will be working on the customer service module and so on. Assuming that we have statistics on transaction volumes from say the previous year, using simple mathematical logics, we can derive a workload model. But first lets define some variables:

Total application concurrency – C

Concurrency of a module y – Cy

Total number of modules in the application – m

Therefore, $C = C1 + C2 + ….. + Cm$. For sake of simplicity, lets represent it by SUM[C1:Cm]

Now lets get into distribution within a module. Lets say the total number of transactions in the module y is n. Consider a transaction x in the module y. Lets say the target volume of x is Vx per hour and the length of x is Lx.

Its important to note that the target transaction volumes should be of the time of the rollout. So, if the requirements were defined in 2016, the application goes live in 2017 and the growth rate is 10% then the target volumes for performance testing should be 120% of the 2016 volumes.

Since each transaction has its own length, i.e. a different number of pages, it is important to first translate business volumes into page views and then go over distribution. Hence, the target page views per second, i.e. $Tx = Vx * Lx$

User distribution i.e. the distribution of the module level concurrency amongst its transactions or use cases will be a function of the target page views Tx.

Therefore, concurrency of a transaction x in a module y i.e.

$Cx = ROUND (Tx / SUM[T1:Tn]) * Cy$

The excel above is a sample workload model for our example. Please note that the values are mere assumptions and in no way represent the actual volumes of bank.
The information at hand was the distribution of a concurrency of 604 users across the three modules, Teller Banking, Online Banking and Phone Banking. Also known were the target volumes for each of the shortlisted use cases. A study of the use case navigation and call flow helped determine the length (number of pages) of each use case. Applying the above formulae over the given information, targeted throughout (Page Views per second) per use case and a concurrency distribution within each module was calculated.

Once created, it is important to get a sign-off on a workload model before starting execution. This ensures that the requirements set forth for the Performance testing exercise are correct and validates the assumptions made.
Since a workload model relates more to the business, it is important to represent the information well. A pictorial representation of information is more likely to be well noticed and understood when compared to an excel containing a whole lot of numbers.

4. WORKLOAD VALIDATION:

In order to redesign the complete workload model it is recommended to do an early validation such as reverse calculation, Think time between pages(TT), Avg response time for each page(RT), time to complete execution x-, Achieved throughput

Requirement analysis, market research and solution design are based on a series of assumptions and it is important to ensure that the assumptions are correct by validating that the goals are achievable. This validation can be done without having to execute the load tests, just by doing some reverse calculations.
For example, lets assume that the average Think time between pages i.e. TT is set at an average of 10Secs and the Response Time target for each web page i.e. RT is 4Secs.
Hence the throughput of a business transaction x that can be achieved by the derived Concurrency Cx is
$Vx = Cx * 3600 / (Lx * (TT+RT))$
where Lx is the length of x i.e. number of pages. If the achieved Vx is in line with the targeted business transaction volumes then it is safe to say that the assumption of think times and the response time requirements are correct.
Validation can also be done post-execution at either the front-end or the back-end. At the front end, there are load generation tools that report counts of execution of transactions under test. Lets take the example of the IBM Rational Performance Tester load test tool. In the test report as one of the metrics, you can see the number of hits made to each page in the test suite. This number is a count of how many transactions were successfully completed on the system.
From the back end, post every test run a simple query on the database can give a count of volumes achieved during a test run.

5. THE BIRDS EYE VIEW:

For Performance Testing to reveal accurate characteristics of a system, the workload model should be a close representation of real world production load pattern. For complex enterprise applications user interface is just one entry point into the system. There are several other interfaces, WebServices scheduled jobs etc that share the system resources. To simulate a real world production load pattern it is essential to look at the complete picture and account for at least incoming load from all possible sources.
With the increasing complexity of business models and interdependence on business partners and service providers, interaction with external subsystems through interfaces and exposed WebServices and messaging interfaces is one primary source of incoming load. Other sources are inter-module communications between modules under test and those that are out of scope of the Performance test exercise.
Another activity to account for is the daily Batch jobs and schedulers that run during the regular business hours. For those that run during off-peak hours, its important to test and ensure that the execution of all scheduled batch jobs complete during the designated window and do not overflow into the regular business hours. Along similar lines, there are regular backup and archival activities that need to be allocated resources.
One other consideration that needs to go into a completing a workload is the recurring business activities that take place over and above the regular tasks. For example Close-of-Business, End-of-Month reporting, Quarterly reports etc.

6. SEASONAL WORKLOAD MODELS:

These models are business critical.There are a few domains that every so often, experience a substantial variation in their load pattern. These are called seasonal workloads. For applications that cater to these domains, ensuring performance and stability during such seasonal workloads also becomes the responsibility of the performance test exercise. Some examples of such seasonal workloads are:
•eCommerce Applications for Retailers: End of Season Sales, Holidays like Christmas and Thanksgiving
•Banking and Financial Applications: End of Year Closing
•Job Portals: Graduation Period
•Human Resource Management Systems: Appraisals etc

7. SELECTIVE MODELLING – RISK ANALYSIS:

There is always some amount of risk involved with selective modeling. Some transaction, some piece of code, SQL, stored procedure etc always rolls out without being performance tested.

An untested transaction can consume excessive system resources, starving other transactions of computational resources and causing a delay in overall system responses, or in the worst case scenario, crash the system.

However small, this risk associated with selective modeling can raise several flags if it has the potential to cause loss of revenue for the business. Because it is highly impractical to load test every transaction, a mitigation strategy needs to be defined.

There is no one thing that can be done to ensure that the system is risk free from performance problems. Several efforts have to run in parallel to cover maximum ground.

•Use Functional tests, UAT and System tests to detect bad transactions

•Monitor servers during UAT and Functional tests

•Load the test environments with near-production volume data

•Analyze offline reports from test servers for any abnormal system usage

•Plan one round of Performance testing with UAT or Functional tests running in parallel on the same environment

Tips:

These tips are some lessons that have been learnt from requirement gathering processes with several customer and hence are generic and applicable to all domains like banking, insurance, retail, telecommunication etc:

•Make sure you have a complete understanding of how the business that is being served by the application. What major functionalities does it cater to and what external systems does it interact with. Try to relate that to the solution design

•If and when possible, visit the business on-site to understand the system usage and study the load patterns

•Always set targets at peaks and not the average volumes

•Account for growth rates by targeting the volumes projected for the rollout timeline

•For a new system with no existing data, derive the data volumes. During execution, load test with databases holding at least near-production volume of data

•Ensure that there is room for server maintenance activities at average loads

•Last but the most important, get a sign-off on the requirements set forth for performance before starting execution

CONCLUSION:

In this session we have gone over the process of gathering and defining requirements for performance testing of enterprise applications. We have seen how workload models can be derived for simple as well as complex use cases using the data available from various sources on projects.

We listed some factors that can help in defining the scope of performance testing activities, the risks involved and possible mitigations for addressing business concerns arising from not performance testing all transactions.

In conclusion, there is no one defined method for creating a comprehensive workload model. The selection process has to be a factor of business priorities, application complexity and project timelines. While there is always some amount of risk involved with performance testing over selective modeling, a lot can be done to mitigate or minimize the possible impact on business.

REFERENCES:

- **AWStats**
- **Google Analytics**