Managing Scalability, Availability, and Performance Requirements for Cloud Services

Rodolfo Kohn[1]

[1] Intel Corporation, ASDC, Cordoba, Argentina
rodolfo.kohn@intel.com

When developing services software meant to reach millions of users through the Internet, unlimited scalability, high performance, and 100% availability turn essential qualities that will support, or seriously disrupt, business growth. This is probably the most important challenge for the new software and infrastructure paradigms in the cloud. This paper proposes a combination of measurable and non-measurable requirements, specific, usually overlooked, process activities, assets and guidelines to manage these software qualities from the early product conception and then throughout the whole product lifecycle. It discusses difficulties found incorporating these practices into the software development process in real-world projects as well as their solutions. Furthermore, concrete results and lessons learnt are presented showing advantages and drawbacks of the approach.

Keywords: Scalability, Performance, Availability, Requirements, Cloud, Services, Non-functional Requirements, Software Qualities.

1 Introduction

1.1 Problem Statement

If functional requirements define system behavior, non-functional requirements (NFR's), a.k.a. qualities, determine how well the system behaves. Nonetheless, it is difficult, if not impossible, to specify non-functional requirements in a SMART way, to track them, to arrive to the best solution, and to assure they are satisfied. For this reason, developers and managers tend to push their consideration as late as possible in the development cycle. As a consequence, often problems are discovered too late, in production when customers have already been affected, or, at best, during system test a few days before planned release. Usually, NFR's are treated as soft goals, i.e., goals that need to be addressed not absolutely but in a good-enough sense [1] and probably this is what makes them so difficult to manage.

Performance, scalability, and availability are challenging system qualities to drive during the software development process and even more during the whole product lifecycle.

This is due to various factors including the difficulties inherent to all non-functional requirements. First, it is usually impossible to specify these requirements in an absolute and measurable way [1]. Second, there is high level of subjectivity when evaluating these qualities. Third, it is always possible to improve performance and scalability for a certain cost: there is a constant trade-off between good-enough and cost. Fourth, it is expensive to validate these requirements due to a tight relationship with the environment and the large spectrum of workload combinations. Fifth, it is impossible to validate these requirements in an absolute way: you just reduce risk margins.

Probably for lack of experience, it is difficult to convince managers to invest resources to avoid future problems while weighing against giving up required functionality. The same occurs with the developers pressed to implement more functionalities in a shorter time. Finally, performance and scalability acceptance criteria usually are not adopted for release.

However, new business models brought by the Internet, technology progress, and hardware commoditization have changed the panorama.

On one side, Internet enables the possibility to reach millions of users on a 7x24 basis with no need to invest in immense capital. Whereas this is wonderful for business, it's a real challenge for software engineers and IT administrators. Indeed, software products have to endure greater, and highly variable, workloads and availability turns essential to keep business growing. Software will undergo stringent conditions that will bring out every "unexpected" and hidden error.

On the other side, technology advances have made hardware more accessible thus allowing more small companies to buy devices like load balancers, adopt disaster recovery solutions, or use Platform-As-A-Service capability, such as provided by Amazon, Google, Microsoft Azure, etc. As a consequence, there is no reason for enterprise software not to adopt high availability concepts and not to exploit their benefits. For software engineers this implies that software issues will have greater impact.

In this new world of cloud services, performance, scalability, and availability are essential for business success:

- When reaching millions of users in different regions, systems have to deal with unpredictable workloads. Even though a system may bear a light workload at the beginning, if business is successful a dramatic load growth may suddenly occur. The service must be able to adapt and cope with this new load transparently for end users. However, each service may have a different rate of adoption, and sometimes there will be little advance ramp-up before an exponential spike in demand.
- In a competitive and brave world, if users have problems accessing a web site or a service, either for unavailability or for slow response time, they immediately move to a different provider. Not to mention thousands of

blogs, newspapers, and rest of the media, eager to wreak havoc when well known companies are involved.

- Additionally systems are going to suffer all kinds of denial-of-service attacks and they must be able to deal with them without affecting availability.

1.2 Proposed Solution

Dealing with nonfunctional requirements is not a new problem and various studies proposed distinct approaches to handle them. For example, [2] proposes a process-oriented approach to achieve non-functional goals. In [3] a performance engineering methodology based on UML is proposed. However, these works take assumptions that in practice are not always available. For example, in [2], an input for the process is the performance goals and most of the times defining performance goals is the problem because nobody has defined them and teams have difficult discussions to reach an agreement and the disagreements are revived when goals are not achieved. The solution proposed here does not compete with the mentioned methodologies but it consists of a higher level pragmatic approach intending to influence the software development process. It involves a holistic approach that is not limited to just measurable requirements but also to specifications that are not measurable but guide the software development and validation. It also includes a set of process activities and assets that assure the team is continuously considering the performance and scalability needs, not specified as measurable requirements, throughout the development process and when product is running in production. Moreover, architectural, coding, and testing best practices should be part of this solution. This approach was adopted for our projects and it is called SCAP process. This process may contain methodologies as the ones referred above and any performance engineering practice.

2 SCAP Process

SCAP Process created for our projects requires the following activities:

- Workload estimations at the Requirements Gathering phase and more detailed estimations after architecture and design definition
- Performance, scalability, stress, and longevity tests after certain development sprints and before every release
- Creation of performance baselines
- Performance bottleneck analysis when performance targets are not reached, when system does not scale as expected (less than 70% improvement) or when performance is to be improved
- Capacity planning based on performance tests results , experience, and business forecasts

- Before delivering software to the hosting environment, a report per application is generated characterizing it: services supported, transactions per application, detailing components involved per transaction, classification and estimated number of SQL transactions in the database, messages sent back and forth per transaction, estimated HTTP payload length in Bytes per message, expected workload (average, normal spikes, and eventual bursts)
- Monitoring in production: internal monitoring -controlling measurements against thresholds and gathering workload feedback- and external monitoring of availability and performance

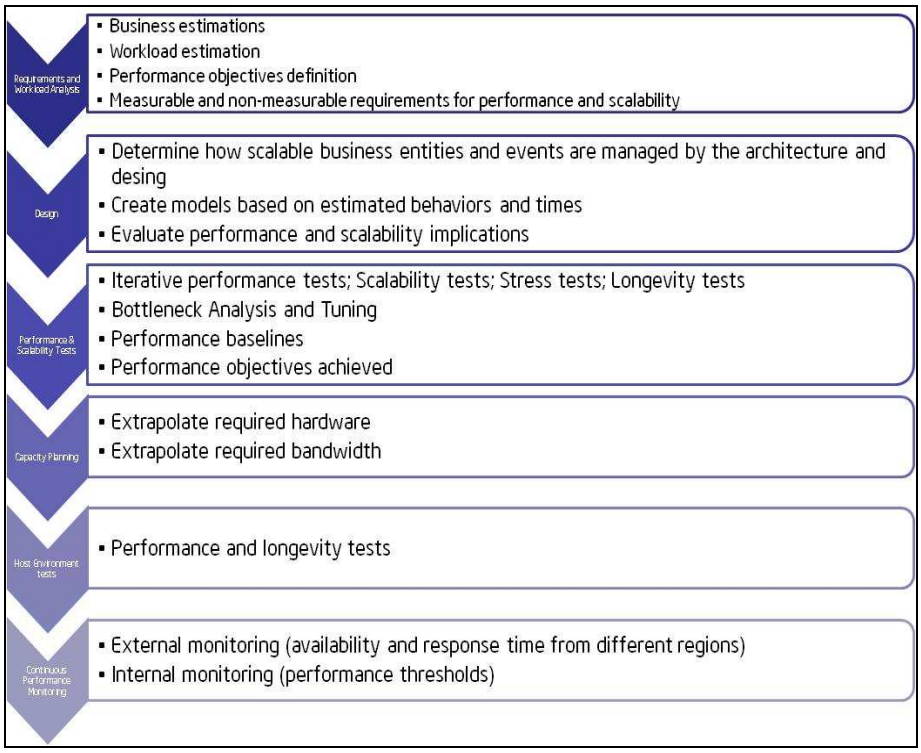

Fig. 1. SCAP process activities.

3 Requirements

It is impossible to specify measurable requirements that will assure performance, scalability, and availability by themselves. However, by implementing certain requirements it is possible to reduce risks of failing in providing the product with these qualities in a good enough way.

It might be argued that certain requirements go too far into design decisions. However, it is also true that if software is not architected and designed based on certain rules, they will not be able to have unlimited scalability, performance will be compromised under load peaks, and availability targets will be in serious risk.

3.1 Scalability Requirements

Measurable scalability requirements should include the following list:

- The system shall be able to horizontally scale in all layers including data storage [4]
- All software components in all layers shall be replicable with different instances running in different servers, either virtual of physical
- The system shall be able to horizontally scale both by adding or removing boxes. It shall not fail upon replication or when a replicated instance is removed
- Given a scale unit formed by a set of servers at all layers. Adding a second scale unit shall yield a minimum of 70% performance increase. This is just an industry rule of thumb and could be higher. If performance had little increment, say 10%, by adding a new server, the system is considered to have poor scalability, probably due to a poor architecture or design. Ideally, duplicating a scale unit shall yield 100% performance improvement. However this is not always possible as bottleneck factors vary.

3.2 Performance Requirements

Measurable performance requirements should be stated as performance targets. Performance targets can be specified as:

- Response time (e.g. end-to-end transaction, transaction between web server and web service server)
- Throughput (e.g. requests per second, transactions per second, etc.)
- Resource utilization (e.g. processor utilization, power consumption, disk access, network utilization)

For backend systems, you should always define response time targets and throughput targets. The question that immediately arises is how to define the correct performance targets. This can be a contentious issue when performance targets are not achieved.

Performance requirements usually are not fixed as they can continuously change based on business progress. This is especially true for throughput. So here it is recommended to follow a workload analysis and estimation activity to set throughput targets based on business estimations and system architecture. It is also necessary to review them with certain periodicity, e.g. quarterly.

The throughput target specification should follow the following steps:

- Identify scalable business entities: it could be users, units of products delivered (applications, cars, etc.), providers, etc.

- Identify scalable business events: purchases, subscriptions, deposits, etc.
- Obtain forecasts of business entities and events: it is desirable to obtain them from business groups but if it is not possible a best guess should be done based on available data and it should have consensus among all stekeholders.
- Estimate behaviors that will trigger business events and determine average of event occurrence per unit of time. For example, 50% of users purchase on weekends. Marketing professionals input would be highly valuable in this task.
- It is a rule in IT that systems must not be planned for average but for peak load. A rule of thumb tells that plans have to be done for average multiplied by 4. Alternatively, another heuristics can be applied, using Poisson distribution for example. In any case it is recommended to keep simple rules and automated to make the process repetitive; otherwise it will be too expensive.
- Finally, throughput targets per request, transaction, or operation, are derived from the rules applied.
- These steps should be repeated with certain periodicity, e.g. quarterly.

In the case of response time, other rules apply because it depends on user experience. The objective is to keep the user happy or at least not to disappoint them. Usually, response time targets are more stable than throughput.

3.3 Logging and Monitoring Requirements

Logging is especially important to keep availability. When an error arises, it is essential to detect it and find out the root cause as fast as possible. Logs must keep a standard structure, like CIM [5] or ITU X.733, in order to enable automatic processing for detecting errors in real time and for helping load trends analysis.

Monitoring is essential in production to detect faults or other events in real time and react as fast as possible to fix problems. The data gathered through monitoring can also be used to analyze load trends, understand workloads and compare them with estimations. Thus it is possible to react to load trends before system is overwhelmed.

Logging and monitoring requirements contribute to keep high availability.

3.4 Availability Requirements

Certain measurable requirements will be necessary to improve availability:

- If a component is replicated failure of one instance shall not affect other instances
- Every component shall be stateless and any state that needs to be maintained shall be stored in a data storage mechanism accessible by all components.
- All components shall execute without change when a load balancer is added
- For all applications, a level of criticality shall be defined. This is going to be useful when defining SLA's

- Every component shall be assigned a level of criticality based on the applications it supports: some components may not be critical and SLA could be less stringent
- The level of infrastructure availability shall determine the application availability. An application should not reduce availability
- Application upgrades shall not require service interruption
- The application shall gracefully behave during component or data center failover handling it transparently for end-users.

3.5 Non-measurable Requirements

Non-measurable requirements will help understand the order of magnitude to which business entities and events should be able to scale. These are useful to guide architects and developers when the application architecture is defined and components are designed. They are also useful to guide QA engineers when designing workloads and preloaded data for performance test.

During analysis, design, and coding there are constant decisions of trade-off among different qualities: maintainability, flexibility, performance, security, etc. Understanding and entity must scale to millions will push to simpler data models or specific, sometimes difficult to maintain, shortcuts when dealing with these entities. Alternatively, it could lead to smarter decisions of meta-language generation that allows the developers to deal with maintainable scripts that is later transformed into high-performance code.

Examples of recommended non-measurable requirements we used:

- The system shall be able to provide 9.99% availability (3 nines).
- The system shall be able to handle hundreds of millions of users (which would be different than developing for thousands of users)

4 Architecture and Design Guidelines

Architecture, design, and coding guidelines help the team keep focus on best known practices that help systems keep good enough performance and scale horizontally with acceptable performance increase.

Developing for parallel processing, stateless components, horizontal scalability even in the data layer, client-centric eventual data consistency [6], programming for failures are concepts required for cloud applications.

5 Results

Our product provides common services like live update, provisioning, events, and others to specific applications intended to reach hundreds of millions of users and devices.

This combination of measurable and non-measurable requirements, process and guidelines within the SCAP process helped the team create a scalable product that can keep up with our expected business growth during this and the following years.

After more than a year in production, our system maintained 100% availability and there was instant notice of any minor incident like detection of response time decrease by the external monitoring system at certain moments.

Regarding QA, four types of performance tests were designed:

- Performance tests
- Stress tests
- Scalability tests
- Longevity tests

Based on workload analysis and estimations, QA team was able to create representative workloads to test the application. It is also possible to continuously improve the testing workloads based on information obtained from production and discussions with the development team.

There is complete awareness in the teams about these requirements. This not only helps the product development but also enriches reviews output and promotes discussions between teams when issues are found.

5.1 Performance Test

As mentioned before, four types of performance tests are executed. Performance tests help determine whether the system can achieve its performance targets with the existing hardware. Stress tests are intended to find the system break points and discover weird errors like memory leaks, race conditions, deadlocks, etc. Scalability tests are intended to understand application scalability profile and also to find the best ratios among servers in different layers. By determining the optimum ratios, it is possible to define a scale unit for the system. Longevity tests are meant to understand system behavior running at average load during long periods of time.

All of these tests are necessary for services and the success of these tests implies there are more possibilities to achieve good enough service availability and there is less risk of violating established SLA's.

The results of stress and performance tests refer both to software and hardware capacity. This means that usually performance can be increased by software modifications or by adding servers.

Scalability tests demonstrate how by adding servers (horizontal scalability), capacity and performance are increased. Its results usually depend on the software and system configuration. However, if a bottleneck is found in the DB, it can be improved by scaling up (adding hardware to the DB server). DB horizontal scalability, by replication or partitioning, may require application re-architecture.

Based on the workload analysis and workload estimations, QA team was able to design representative workloads to test the system. Then, for each version, we are able to have a good idea of how our system would behave under those workloads and determine whether our system can achieve performance targets or not. Additionally, we can measure how new features affect performance baselines. Results usually open

the way to discussions within the team that allow managers taking release decisions with richer data. Additionally, test results are useful to identify further opportunities for improvement.

5.2 Performance and Scalability Results

In different scalability tests our product demonstrated a great level of scalability. At some point in the time we noted, as expected, performance increased 1x per new server added at the application layer. We needed to execute a special test borrowing a huge number of servers to reach our next scalability inflection point. Figure 2 shows the results of scalability tests executed beyond a baseline called 1x, with N servers. It was not until 10x that we were able to find our inflection point where the scalability curve begins to clearly separate from linear scalability. For confidentiality reasons, numbers have been normalized. After we found and solve the bottleneck, our product was able to have even better scalability.

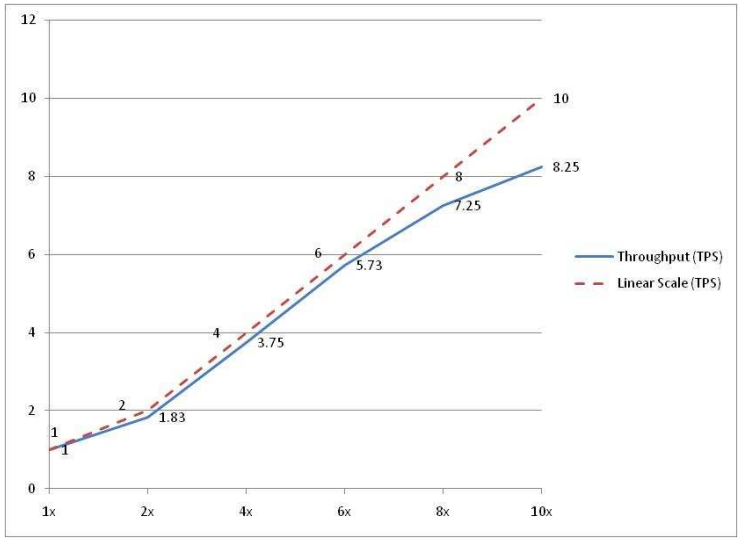

Fig. 2. System scalability results beyond a selected baseline of 1x.

Since we begin to execute SCAP process our application performance in a minimum scale unit increased more than 10x. We were able to control our performance and continuously improve it. Having more performance per scale unit helped us reduce capital costs.

Today our application can scale out to deal with our expected loads during the following 2 years.

5.3 Availability Results

As mentioned above, we measured 100% availability during more than one year. This system quality not only depends on software but also on infrastructure. However, software should not affect negatively the availability provided by the infrastructure.

5.4 Discussions on Performance Targets and Tests Results

SCAP process improved the quality of team discussions regarding system performance, availability, and scalability by providing data and focusing the discussions.

First of all, the process makes it possible to reach consensus on what performance targets are for each application and under specific workloads. Different stakeholders can focus their discussion on how they think a business entity will generate a business event. Once, they agreed on that, performance target calculation is straightaway with a well known and accepted formula. Whenever a target is not achieved, a meeting is called to discuss about the test, the results, and how they may affect the business while that software version is in production. Then decision is made about approving or not a release. Moreover, at that point there are no discussions about validity of performance targets.

It is also possible to understand the effect of changes between versions. It is possible to detect lower performance trends and take actions for next releases.

Performance test results helps managers and technical leaders identify bottlenecks and evaluate whether investing effort in solving it and when.

Finally, and very important, it is possible to provide IT with concrete data when it must be decided about the hardware to invest in production. This situation permits to save costs on spare capacity that may be installed and not utilized while at the same time minimizing risks of not having enough capacity to deal with peaks and bursts.

5.5 Consequences in the Cloud

Having data about workload analysis, system capacity, scalability profile, and real workload in production was important to define a fix capacity base for a system and estimate future needs for burst capacity. This was useful to determine future budget required for operations and take decisions whether the revenue will pay off.

Having the scalability profile was necessary to obtain the optimum scale unit for the application under certain workloads.

5.6 Opportunities for Improvement

After running this process for more than one year we found many opportunities for improvement. The most important was the cost of executing these tests. We were continuously progressing in automation in order to be able to run these tests with lower costs and to integrate them into our agile development process. This will allow

us to find issues earlier in the development cycle providing continuous input about how a new feature is affecting.

Regarding workload analysis, it might be argued that there is a high level of guessing when estimated workloads. However, this is also true when estimating development effort and this has proved been a great practice. Additionally, as we have more information from production, workload estimations become more accurate.

5 Conclusions

This paper proposes a holistic approach to manage specific system qualities: performance, scalability, and availability. A process, SCAP, is proposed to combine measurable requirements, non-measurable requirements, activities and assets, and architecture guidelines to manage these system qualities. This process is not an alternative for performance engineering practices and methodologies to handle non-functional requirements. It is a higher level approach that can contain them.

Disaster recovery and geographical distribution with multiple availability zones were not discussed in this paper in order to reduce the scope but achieving the mentioned requirements is a pre-condition for these capabilities.

Here, it is preferred to discuss about managing performance and availability instead of just predicting or evaluating it because it is a more comprehensive term that encompasses prediction, generation of input to the product architecture and design, generation of input for capacity planning, evaluation, reporting, and monitoring.

It may be argued that results are not a direct consequence of this process but just what skillful developers did. However, the continuous focus on scalability, performance, and availability, the discussions around workload estimations and issues found during performance tests, the awareness of possible problems after workload estimations, have created an environment where there is continuous improvement of performance and scalability. Additionally, identified bottlenecks and errors helped the team invest the effort in solutions that would yield highest improvement.

As of today, this process proved to be useful to avoid problems in production even when eventual bursts occurred. As it was explained, our system demonstrated to be highly scalable and availability was 100% all the time.

References

1. L. Chung, J. C. Sampaio do Prado Leite, “On Non-Functional Requirements in Software Engineering,” Book Chapter, p.363-379, “Conceptual Modeling: Foundations and Applications,” Springer Berlin / Heidelberg, 2009
2. Mylopoulos, J.; Chung, L.; Nixon, B.; , "Representing and using nonfunctional requirements: a process-oriented approach," Software Engineering, IEEE Transactions on , vol.18, no.6, pp.483-497, Jun 1992
3. Andrew J. Bennett and A. J. Field. 2004. Performance Engineering with the UML Profile for Schedulability, Performance and Time: A Case Study. In Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and

Simulation of Computer and Telecommunications Systems (MASCOTS '04). IEEE Computer Society, Washington, DC, USA, 67-75.
4. Michael, M.; Moreira, J.E.; Shiloach, D.; Wisniewski, R.W.; "Scale-up x Scale-out: A Case Study using Nutch/Lucene," Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, vol., no., pp.1-8, 26-30 March 2007
5. CIM, Common Information Model, website: http://www.dmtf.org/standards/cim
6. A. Tanenbaum and M. van Steen, “Distributed Systems, Principles and Paradigms,” Prentice Hall, 2002