On Preventively Minimizing the Performance Impact of Black Swans

Vision Paper

Andre B. Bondi*
Software Performance and Scalability Consulting LLC
Red Bank, NJ
http://andrebbondi.com
andrebbondi@gmail.com

ABSTRACT

Recent episodes of web overloads suggest the need to test system performance under loads that reflect extreme variations in usage patterns well outside normal anticipated ranges. These loads are sometimes expected or even scheduled. Examples of expected loads include surges in transactions or request submission when popular rock concert tickets go on sale, when the deadline for the submission of census forms approaches, and when a desperate population is attempting to sign up for a vaccination during a pandemic. Examples of unexpected loads are the surge in unemployment benefit applications in many US states with the onset of COVID19 lockdowns and repeated queries about the geographic distribution of signatories on the U.K. Parliament's petition website prior to a Brexit vote in 2019. We will consider software performance ramifications of these examples and the architectural questions they raise. We discuss how modeling and performance testing and known processes for evaluating architectures and designs can be used to identify potential performance issues that would be caused by sudden increases in load or changes in load patterns.

CCS CONCEPTS

• **Software and its engineering → Software performance**.

KEYWORDS

Performance testing; performance measurment; software architecture; load modeling; black swans

ACM Reference Format:
Andre B. Bondi. 2021. On Preventively Minimizing the Performance Impact of Black Swans : Vision Paper. In *Companion of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21 Companion), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3447545.3451204

*Also with Stevens Institute of Technology.

ICPE '21 Companion, April 19–23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8331-8/21/04…$15.00
https://doi.org/10.1145/3447545.3451204

1 INTRODUCTION

Disasters, changes in social or political conditions, deadlines, and the transient availability on line of scarce and highly valued commodities such as cheap plane tickets, concert tickets, or an appointment to get a COVID 19 vaccination all trigger changes or surges in on line activity with accompanying degradations in performance, reliability, and other qualities of service. The surges in traffic associated with deadlines and attempts to obtain scarce commodities are foreseeable. Some changes in use case invocation frequencies due to political and social conditions are not. The ones that are not foreseen are sometimes called black swans, after a term coined by Taleb [10]. Whether or not traffic changes are foreseen, the resultant system stress may expose system flaws that should have been prevented by architectural evaluations and sound design and coding practices. The symptoms of these overloads have occurred in multiple private sector and public sector systems. They are sometimes foreseeable and often preventable, but not always prevented. Moreover, since performance and system tests frequently only cover traffic patterns specified in performance requirements (if such requirements have even been documented), they will not be uncovered before a system is deployed and accessible to the public. Among the challenges performance engineers face in detecting and preventing these problems are the design and execution of load and stress tests that will reveal problems arising from traffic patterns that might be regarded as extreme, persuading management that these tests are necessary, and architecting systems for load scalability and reviewing the designs and architectures to prevent the problems from occurring in the first place[2].

After describing well known examples of performance issues and other symptoms arising from changes in usage patterns, both foreseen and unforeseen, we shall explore ways of preventing the symptoms based on model-driven performance testing and on systematic reviews of the designs and information flows.

The author has no direct knowledge of the design or internals of any of the systems mentioned in this paper. Questions about them are based solely on conjecture, the author's experience, and press reports.

2 SOME WELL KNOWN CASES OF OVERLOAD

Apart from sluggish response times and outright crashes, symptoms of overload include losses or corruption of users' stored data, presentation of available items only to be told they are not available once they are selected, and other failures to complete transactions.

In March 2019, the U.K. parliamentary grievance petition website crashed repeatedly during a campaign gather signatures on a petition to stop Brexit [6]. This was an example of overload caused by an unexpected change in usage patterns. Interestingly, it was not the arrival rate of petition signatures that was causing performance issues, but repeated individual requests to produce and display a map showing the current, up-to-the-minute geographical distribution of the signatories. Users could make this request on demand, apparently generating a massive query on a database that was being updated with each signature. Once the system was configured to redisplay the map every half hour instead of on demand, the performance problems abated [1].

In 2020, the web sites used to claim unemployment benefits in many American states could not cope with the sudden overloads triggered by massive layoffs due to the COVID19 pandemic. These web sites are administered by the states, not by the Federal government. According to press reports [4] [5], the New Jersey unemployment benefits website could not easily be scaled up to cope with the sudden increase in demand. Its business logic is implemented in COBOL, a legacy programming language which few new programmers know. Many of those who wrote the COBOL code have retired. Symptoms of overload included long response times on the web site, back office turnaround of initial claims taking months, and malfunctions including the loss, corruption, or overlooking of fields in claimants' records, such as their banking information for the direct deposit of benefits. Correcting the problems caused by loss or corruption of user data required further interactions with the system by both beneficiaries and New Jersey Labor Department staff. The overload situation was only partially mitigated by restricting on-line access by beneficiaries to a few half-hour time slots per week, determined by the last four digits of one's Social Security number.

There are several other examples of websites crashing on overload either because of social interest or because of foreseen deadlines. Among those that should have been expected were:

- In New Jersey and many other states, COVID19 vaccination registration websites have experienced surges that some have likened to tickets coming on sale for a rock concert, with overload symptoms similar to those of many US unemployment websites.
- The Australian census website crashed on evening of the day all residents were supposed to complete their online entries. Unlike the NJ unemployment office, the Australian government did not attempt to stagger the arrivals to spread the load over time.
- The U.K.'s Brexit voter registration website crashed on the last day to register.

Among those overload situations that could not be reasonably expected in advance were:

- The Canadian immigration service's website experienced a surge of visits from the U.S.A. on the night that Donald Trump was elected president. It crashed on overload. [11]
- The news website for CNN crashed on overload on the morning of the 9/11 attacks. In the New York City region, the overload may have been compounded by the reduction in radio and TV broadcasting until transmission could be moved from the World Trader Center to the Empire State Building.

3 GUARDING AGAINST BLACK SWANS IN THE SOFTWARE DEVELOPMENT LIFE CYCLE

3.1 Overview

In this section, we illustrate how one might guard against an abrupt change in load patterns that might constitute a black swan. After discussing methods in general terms, we focus on two examples, the U.K. Parliament's grievance petition website and the New Jersey Department of Labor's unemployment benefits website.

Measures to guard against the performance consequences of black swans should be taken at all phases in the software development life cycle, regardless of whether the development process is based on the waterfall model or an agile method. When the workload consists of a family of use cases or demands, one should conduct an application flow analysis to identify points of contention between use cases, such as hardware resources like CPU, I/O devices, memory, and network bandwidth, and passive software resources such as tables, locks, threads, and JDBC connectors. The application flow analysis should reveal whether user needs can be served by structuring use cases that are resource-intensive so that they are invoked at controlled frequency. It should include a review of configuration settings to ensure that bottlenecks are not needlessly induced by setting the values of tunable parameters, e.g. the sizes of logical object pools, to values that are so low as to impeded the use of hardware resources.

Once potential bottlenecks and their holding times per transaction of each type are identified, the combinations of arrival rates of use case invocations should be varied over a hyperplane and fed into a simple model to determine which points of contention will become bottlenecks and which mixes of arrival rates are likely to cause saturation. These mixes and neighbouring combinations of arrival rates are candidates for load levels to be used in performance testing to see whether saturation will indeed occur. At that point, an architectural tradeoff analysis method (ATAM) could be used to determine proposed feasible architectural and design changes that might mitigate the problem[7].

We now turn to two specific examples. The U.K. parliamentary petition website involves the concurrent and frequent invocations of use cases in a way that causes considerable performance degradation. The New Jersey unemployment benefits web site is an example of an architecture that is hard to scale, in part because of the use of an enduring legacy system that appeared to be difficult to speed up or scale up.

3.2 The U.K. Parliamentary Petition Website

The use cases we consider here are the electronic entry of signatures on a petition, which we call SignPet, and the generation of a map showing the number of signatories by geographical region, which we call MapG. SignPet and MapG could originally be invoked by anyone who accesses the website. For the sake of discussion, let

us suppose that SignPet involves entry of the signer's name, address, e-mail address, and parliamentary constituency, while MapG involves generating counts based on a scan of the entire database table of signatures at the time the request is generated. A database may facilitate the generation of other analytics. Notice that every signature causes the addition of a row to the signature table. The table is an object of contention between the two use cases. Moreover, it would grow over time as signatures were added. This means that the cost of MapG grows over time with this implementation. SignPet's processing cost may grow, too, but perhaps by not as much, because insertion should never involve a table scan. The total number of signatures was about 5.7 million. It is not hard to see that MapG as described is resource intensive when the number of petitioners is large, but that it might not be if the number of petitioners were several orders of magnitude smaller. A designer and a performance engineer could guard against the performance consequences of the Brexit overload scenario by asking if it is possible to maintain separate counters of the number of signatories in each region, and whether it is sufficient for the system to update the map every half hour by invoking MapG, while preventing the general public from invoking it on demand. These small design changes can guard against the worst effects of this particular black swan. We do not know whether the designers addressed this issue before the black swan occurred. A performance engineer could make the case for guarding against a black swan like this one by building a simple model to show which resources would be saturated by its occurrence and the mixes of loads for which saturations are predicted. These should include hardware or virtual hardware resources such as CPU, disc, and memory, and software resources such as database locks. We suggest that every implementation should be preceded by an examination of such a model with hypothetical parameters based on past experience.

3.3 An Unemployment Benefits System that is Difficult to Scale

Based on press reports, we conjecture that the back end of the New Jersey unemployment benefits website was implemented as a monolith in COBOL before the use of replicated instances of business logic became commonplace. The number of newly unemployed in the USA grew to record levels rapidly with the onset of the COVID19 pandemic. The performance of the system in New Jersey had been adequate until then, with short response times at the claims interface and benefits being paid on time. Scaling out might be achieved by replicating the business logic in containers. Transactions would be routed to them with round robin or some other load balancing mechanism. This proposed remedy is based on the assumption that the back end database is more modern and can be more easily scaled if its existing capacity is not sufficient. A modeling exercise could be conducted to help make this determination while assessing tyhe number of needed containers and the size of the platform needed to host them. We do not know whether this was done.

As a workaround, an attempt was made to constrain the peak load at any instant by breaking the user identification (Social Security) number space into approximately 50 groups. Each group has its own set of 30-minute time slots on a few days each week during which its members are permitted to log in. That reduces the extent to which the system has to be scaled to cope with the vastly increased number of weekly claims for benefits. This is a mitigating strategy that, combined with others, seems to be effective, though it does impose some inconvenience on the claimants.

It turns out that the risk that the New Jersey unemployment system would not be able to cope with a large spike in claims was known as early as 2003, and nothing was done about it. Other public sector systems in New Jersey are also known to be at risk for similar reasons[8]. It is clear that the problem might have been headed off by an upgrade to the system that would have made it more robust. This understanding provides a strong case for periodically assessing the risk to the system and the ability of new technologies to mitigate it in a cost-effective manner. Unfortunately, creaky IT infrastructure, whether public or private, is not as visible to the general public as a road, building, or bridge that has fallen into disrepair. The means to determine the extent to which ongoing improvements are needed is well known in the software performance engineering community. Unlike bridge or road repair, the improvements need not cause disruption. The political and public policy aspects of this problem are outside the scope of this paper and will not be addressed further here.

4 CONCLUSION

Current technology enables the replication of systems to make them more robust and scalable in the face of sudden or sustained surges of load to an extent that may not have been feasible with technologies that were prevalent 25 or 30 years or even 40 years ago. We have seen that some of these surges are foreseeable or even scheduled, as is the case with systems that one must use to comply with government obligations by a set date or within a narrow time window. Others are due to unanticipated changes in social conditions or changes in the way the system is used. The methods and practices we have suggested to guard against the undesirable performance impacts of sudden workload changes and surges are well known in the performance community [3][9].

Success at preventing the potential inconvenience associated with disasters and other black swans is not often visible to management or the public at large unless performance degradation or failure and recovery from them are highly visible or unless they have occurred in recent memory. The poor visibility of success should underscore its value rather than diminishing it.

REFERENCES

[1] 2019. Petition to revoke article 50 hits 3.5m signatures. *The Gaurdian* (2019). https://www.theguardian.com/politics/2019/mar/22/petition-to-revoke-article-50-hits-3-million-signatures
[2] Len Bass, Paul Clements, and Rick Kazman. 1998. *Software Architecture in Practice.* Addison-Wesley.
[3] A. B. Bondi. 2014. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice.* Addison-Wesley.
[4] Elena Botella. 2020. Why New Jersey's Unemployment Insurance System Uses a 60-Year-Old Programming Language. *Slate* (2020). https://slate.com/technology/2020/04/new-jersey-unemployment-cobol-coronavirus.html
[5] Jeff Goldman. 2020. Record number of unemployment applications crashed N.J.'s website, Murphy says. *NJ.com* (2020). https://www.nj.com/coronavirus/2020/03/record-number-of-unemployment-applications-crashed-njs-website-murphy-says.html#:~:text=New%20Jersey's%20online%20unemployment%20appl
[6] Jennifer Hassan. 2019. 'Cancel Brexit' petition surpasses 5.7 million signatures, as Parliament agrees to debate it. *Washington Post*

(2019). https://www.washingtonpost.com/world/2019/03/21/can-brexit-be-stopped-people-are-trying-so-hard-that-parliaments-website-is-broken/

[7] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carrier. 1998. The architecture tradeoff analysis method. *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No.98EX193* (1998).

[8] Sophie Nieto-Munoz. 2020. N.J. failed to fix unemployment system for 19 years, records show. Now Murphy pleads patience. *NJ.com* (2020). https://www.nj.com/coronavirus/2020/05/nj-failed-to-fix-unemployment-system-for-19-years-records-show-now-murphy-pleads-patience.html

[9] Connie U. Smith and Lloyd G. Williams. 2002. *Performance Solutions.* Addison-Wesley.

[10] Nassim Nicholas Taleb. 2007. *The Black Swan: The Impact of the Highly Improbable.* Random House.

[11] Elizabeth Weise. 2016. Americans really did crash the Canadian immigration site on Election Day. *USA Today* (2016). https://www.usatoday.com/story/tech/news/2016/11/10/100000-americans-crashed-canadian-immigration-site/93587034/