SOFTWARE PERFORMANCE ANTIPATTERNS: PROBLEM & AN APPROACH

[1]Kapil Kumar, [2] Prof. Anil Kumar Solanki
[1]Research Scholar, [2]Professor and Head
[1] Computer Science & Engineering Department, Bhagwant University, Ajmer, 305004, INDIA
[2] Computer Science & Engineering & Information Technology Department, BIET, Jhansi, 284128, INDIA

Abstract: A pattern may be a common resolution to a drag that happens in many various contexts. Patterns capture skilled data regarding "best practices" in code style during a type that enables that data to be reused and applied within the style of the many differing kinds of code. Antipatterns area unit conceptually the same as patterns in this they document revenant solutions to common style issues. they're referred to as antipatterns as a result of their use (or misuse) produces negative consequences. Antipatterns document common mistakes created throughout code development additionally as their solutions. whereas each pattern and antipatterns is found within the literature, they generally don't expressly contemplate performance consequences. This paper explores antipatterns from a performance perspective.

Keywords: Pattern, code Performance, Antipatterns, code Patterns

I. INTRODUCTION

Pattern may be a common resolution to a drag that happens in many various contexts [Gamma et al. 1995]. It provides a general resolution that will be specialised for a given context. Patterns capture skilled data regarding "best practices" in code style during a type that enables that data to be reused and applied within the style of the many differing kinds of code. Patterns address the matter of "reinventing the wheel." Over the years, code developers have solved basically constant drawback, albeit in numerous contexts, over and once more. a number of these solutions have stood the take a look at of your time whereas others haven't. Patterns capture these evidenced solutions and package them during a manner that enables code styleers to look-up and reprocess the answer in a lot of constant fashion as engineers in alternative fields use design handbooks. the employment of patterns in code development has its roots within the work of Saint Christopher Alexander, associate designer. Alexander developed a pattern language for designing cities and planning the buildings at intervals them [Alexander et al. 1977]. A pattern language may be a assortment of patterns that will be combined to unravel a variety of issues at intervals a given application domain, like design or code development. Alexander's work written a lot of of what was, until then, inexplicit the sphere of design and needed years of expertise to find out.

Patterns are delineated for many totally different classes of code development issues and solutions, together with code design, design, and therefore the code development method itself.

Antipatterns [Brown et al. 1998] area unit conceptually the same as patterns in this they document revenant solutions to common style issues. they're referred to as antipatterns as a result of their use (or misuse) produces negative consequences. Antipatterns document common mistakes created throughout code development additionally as their solutions. so antipatterns tell you what to avoid and the way to repair the matter you discover. Antipatterns area unit refactored (restructured or reorganized) to beat their negative consequences. A refactoring may be a correctness-preserving transformation that improves the standard of the code. Refactoring is also accustomed enhance many various quality attributes of code, including: reusability, maintainability, and, of course, performance. Refactoring is mentioned very well in [Fowler 1999].

II. RELATED WORK:

Antipatterns area unit derived from work on patterns. As noted within the introduction, this work is aimed toward capturing skilled code style data. there's an outsized body of revealed work on patterns together with [Gamma et al. 1995], [Buschmann et al. 1996], and therefore the proceedings of the Pattern Languages of Program style (PLoP) conferences. whereas there's occasional mention of performance concerns within the work on patterns, the principal focus is on alternative quality attributes, like modifiability and maintainability

Antipatterns extend the notion of patterns to capture common style errors and their resolution. the foremost intensive work on this subject is by Brown, et al. [Brown et al. 1998]. Their work, just like the work on patterns but, focuses mainly on quality attributes apart from performance.

III. PROBLEM

The problem with a One-Lane Bridge is that traffic could solely travel in one direction at a time, and, if there are a unit multiple lane of traffic all acquiring parallel, they have to merge and proceed across the bridge; one vehicle at a time. This will increase the time needed to urge a given range of vehicles across the bridge, and may additionally cause long backups.

The code analogy to the One-Lane Bridge may be a purpose within the execution wherever one, or solely a couple of, processes could still execute at the same time. All alternative processes should wait. It ofttimes happens in applications that access a information. Here, a lock ensures that just one method could update the associated portion of the information at a time.

it's going to additionally occur once a group of methods create a synchronous decision to a different process that's not multi-threaded; all of the processes creating synchronous calls should move "crossing the bridge".

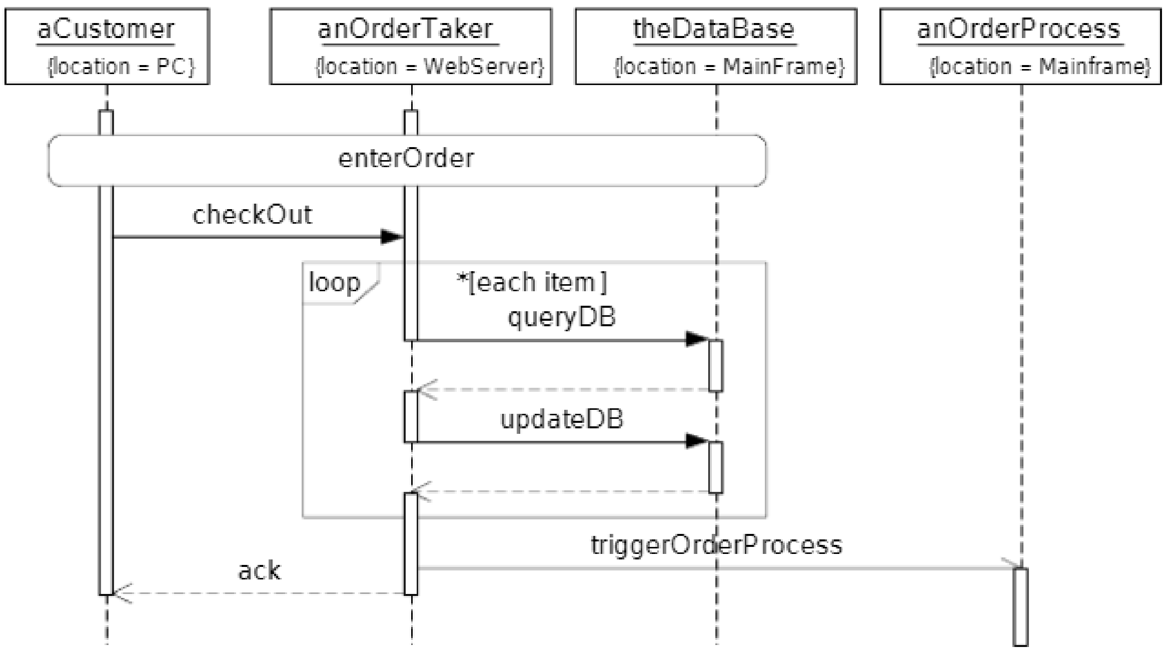

Figure 1: Database Contention Problem

The sequence diagram in Figure 1 illustrates the information variant of the One-Lane Bridge antipattern. every order needs an information update for every item ordered. The structure elect for the information assigns a replacement order item range to every item, and inserts all things at the tip of the table. If each new update should visit constant physical location, and every one new things area unit "inserted," then the update behaves sort of a One-Lane Bridge as a result of just one insert could proceed at a time; all others should wait.

IV. Solution

With traffic, you alleviate the congestion caused by a One-Lane Bridge by constructing multiple lanes, constructing further bridges (or alternative alternatives), or rerouting traffic. The analogous solutions within the information update example on top of would be:

- Use associate rule for assignment new information keys which ends up during a "random" location for Inserts.
- Use multiple tables that will be consolidated later, or
- Use another various like pre-loading "empty" information rows, and choosing a location to update that minimizes conflicts.

Reducing the number of your time needed to cross the bridge additionally helps relieve congestion. a way to try to this can be to seek out alternatives for performing arts the update. The magnitude of the development depends on the intensity of recent item orders, and therefore the service time for performing arts updates.

The relationship is:

$$RT=S/ 1-XS$$

where RT is that the continuance (elapsed time for performing arts the update), S is that the service time for performing arts updates, and X is that the arrival rate.

If you alter the structure of the information in order that you update in multiple locations (so fewer processes sit up for every update), this can be resembling reducing the arrival rate.

V. . Conclusion

Developer will track out problems simply which might cause performance degradation and therefore the major problems is solved by victimization suggestions generated from the code performance analyser throughout the writing section itself. This paper has explored antipattern from a performance perspective. the worth of each antipatterns and their forerunner, patterns, is that they capture skilled code style data. One serious disadvantage of each patterns and antipatterns has been their lack of concentrate on performance problems. The One Lane Bridge drawback is also caused by either information collisions or synchronization delays. a lot of work is required on each pattern and antipatterns that has their impact on performance additionally as alternative quality attributes.

References:

[1] Software Performance Testing Tools – A Comparative Analysis, Mrs. Charmy Patel, Dr. Ravi Gulati, International Journal

of Engineering Research and Development, Volume 3, Issue 9 (September 2012).
[2] Stefan Lessmann, Student Member, IEEE, Bart Baesens, Christophe Mues, and Swantje Pietsch, “Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings”, IEEE Transactions on Software engineering, Vol. 34, no. 4, JULY/AUGUST 2008.
[3] G.Kasi Reddy and Dr.D.Sravan Kumar, “Study on Models of Performance Evaluation of Software Architectures”, International Journal of Computer Engineering & Technology (IJCET), Volume 3, Issue 2, 2012, pp. 378 - 388, ISSN Print: 0976 – 6367, ISSN Online: 0976 – 6375.
[4] The Business Case for Software Performance Engineering Lloyd G. Williams, Ph.D., Connie U. Smith, Ph.D.
[5] Islam, N. and Devarakonda, M, “An Essential Design Pattern for Fault-Tolerant Distributed State Sharing”, Communications of the ACM, vol.39, no. 10, Oct. 1996, pages 65-74.
[6] Smith, C.U. and Williams, L.G., “Best Practices for Software Performance Engineering”. Proc. CMG, Dallas, Dec. 2003.
[7] IEEE Standard 1061-1992. Standard for a Software Quality Metrics Methodology. New York: Institute of Electrical and Electronics Engineers, 1992.
[8] Williams, L.G. and Smith, C. U., “Coping with Performance Anxiety”, http://www.perfeng.com/perfanx.htm, 2004.