On Improving the Dependability of Cloud Applications with Fault-Tolerance

Foutse Khomh
SWAT Lab., DGIGL, Polytechnique Montréal
Québec, Canada
foutse.khomh@polymtl.ca

ABSTRACT

Cloud computing is an increasingly popular paradigm that allows individuals and enterprises to provision and deploy software applications over the Internet. Customers can lease services provided by these "cloud" applications (*a.k.a* cloud apps), ramping up or down the capacity as they need and paying only for what they use. Cloud apps are used in about every industry today; from financial, retail, education, and communications, to manufacturing, utilities and transportation. Forrester Research predicts that cloud apps sales will account for more than 16% of the total software market by 2016. However, cloud apps dependability is still a major issue for both providers and users. Failures of cloud apps generally result in big economic losses as core business activities now rely on them. In this position paper we discuss the current state of the dependability of cloud apps and advocate for the use of fault-tolerance mechanisms to improve the dependability of cloud apps.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Quality—*dependability of cloud applications*

Keywords

Dependability; Cloud applications; Fault-tolerance; Recovery mechanisms.

1. INTRODUCTION

Cloud computing is an increasingly popular paradigm that allows individuals and enterprises to provision and deploy software applications over the Internet. Customers can lease services provided by these "cloud" applications (*a.k.a* cloud apps), ramping up or down the capacity as they need and paying only for what they use. Cloud apps typically run on cloud platforms such as Google App Engine, Windows Azure, or OpenStack. Cloud apps are used in about every industry today; from financial, retail, education, and communications, to manufacturing, utilities, and transportation. Forrester Research predicts that cloud apps sales will account for more than 16% of the total software market by 2016. Yet, the dependability of cloud apps is still a major issue for both providers and users. Failures of cloud apps generally result in big economic losses as core business activities now rely on them [11]. This was the case in December 24, 2012 when a failure of Amazon web services caused an outage of Netflix cloud services for 19 hours. According to Zheng et al. [16], the demand for highly dependable cloud apps has reached unprecedentedly high levels. However, there is still no clear methodology in industry today for developing highly dependable cloud apps. Developers usually delegate dependability issues to the cloud platforms running the apps. A rule of thumb being to deploy the apps across multiple availability zones (AZ) to replicate services, as summarized by the Netflix strategy [15]: *"Deploy in multiple AZ with no extra instances –target autoscale 30-60% until you have 50% headroom for load spikes. Lose an AZ leads to 90% utilization".* Yet, stress tests conducted by Sydney-based researchers [13] have revealed that infrastructure and platform services offered by big players like Amazon, Google and Microsoft suffer from regular performance and availability issues due to service overload, hardware failures, software errors and, operator errors. The response times of these services was found to vary by a factor of twenty depending on the time of day. Moreover, because of the constantly increasing complexity of cloud apps and because developers have little control over the execution environment of these applications, it is exceedingly difficult to develop fault-free cloud apps. Therefore, cloud apps should be robust to failures if they are to be highly dependable.

One way to make cloud apps robust to failures, is to use fault-tolerance techniques. A large number of fault-tolerance techniques have been proposed by the reliability community over the years. Among these techniques we can quote (the major ones) N-Version Programming (NVP) [1], recovery block [5], and N self-checking programming [8]. Unfortunately, because of the cost of their implementations (fault-tolerance techniques require the development and the maintenance of multiple redundant components); the use of these fault-tolerance techniques has been limited to critical systems only, *e.g.*, airplane flight, space/military systems, or nuclear systems. This situation is about to change, as the multiple redundant services and resources that are available in cloud environments offer the possibility to build redundancies in cloud based apps at low costs. Hundreds

http://dx.doi.org/10.1145/2578128.2578228
WICSA '14, April 07 - 11 2014, Sydney, NSW, Australia

of new cloud Application Programming Interfaces (API)s are sprouting up monthly, as tracked by the website ProgrammableWeb [1], which features more than 10,000 APIs today. These service-level APIs provide a lot of redundant services that can be incorporated in cloud apps to implement fault-tolerance. Using patterns like Heartbeat or Watchdog [5], a cloud app can monitor a specific service on which it depends and, in case of failure of this service, redirect requests to a backup service and trigger a recovery mechanism to maintain high availability.

In this position paper, we discuss some recent research on the dependability of cloud apps and advocate for the use of fault-tolerance mechanisms to improve the dependability of cloud apps. In particular, we discuss some challenges related to the implementation of fault-tolerance mechanisms in cloud apps and outline some future research directions.

The rest of this paper is organized as follows: Section 2 introduces some related works on the dependability of cloud apps. Section 3 discusses the implementation of fault-tolerance in cloud apps. Finally, Section 4 concludes the paper

2. RELATED WORK

Recent research studies have proposed software rejuvenation and failure recovery techniques based on virtualization to improve the dependability of cloud apps. Software rejuvenation is a proactive fault management technique aimed at cleaning up the internal state of a system to prevent the occurrence of severe failures due to software aging or transient failures. For instance, Kourai and Chiba [7] propose a technique called warm-VM reboot for fast rejuvenation of Virtual Machine Monitors (VMMs) that enables an efficient reboot of VMM by suspending and resuming VMs without accessing the memory images. The technique, which is based on two mechanisms, on-memory suspend/resume of VMs and quick reload of VMMs, is claimed to reduce downtime and prevent performance degradation due to cache misses after the reboot. Trivedi et al. [12] propose stochastic models that help to detect software aging and determine optimal times to perform rejuvenation. Cully et al. [3] propose a fail-over technique based on asynchronous VM replication. The technique can propagate changed state to a backup host asynchronously at frequencies as high as forty times a second. They use speculative execution to concurrently run the active VM slightly ahead of the replicated system state. In case of failure, automatic fail-over with only seconds of downtime is provided while preserving host state such as active network connections. Israel and Raz [6] modelled the VM Recovery Problem (VMRP) as a formal optimization problem that provides an optimal recovery solution taking into account both the cost of keeping machines active as well as the VM recovery cost, and observed that it is NP-hard. They modelled the problem as an offline optimization problem and proposed a bi-criteria approximation algorithm for it. They also proposed a greedy approach which they claim can reduce the overall recovery costs by 10-15% when compared to currently used approaches.

In this position paper, we adopt the perspective of Recovery Oriented Computing [10] that software faults are facts to be coped with and argue for the containment of faults at the application level. Similar goals have been pursued by [4, 9] which propose fault-tolerance mechanisms for web services. However, because of the dynamic nature of the cloud environment and its various underlying technologies, these fault-tolerance mechanisms cannot be simply transfer to cloud apps. Existing fault-tolerance mechanisms assume that a fixed amount of hardware and system resources is bounded statically to the fault tolerant structures [2]. It is thus difficult to test their effectiveness in a cloud environment, where resources can be provisioned on demand and where the access to resources is regulated by contracts. In the cloud, developers cannot go beyond what is specified in a service contract, neither can they tune, adjust, or modify cloud resources beyond the specifications of their contracts. Also, before implementing a fault tolerant structure in a cloud app, developers must determine the right granularity at which the fault should be contained to minimize recovery time and costs.

3. IMPLEMENTATION OF FAULT TOLERANCE MECHANISMS IN CLOUD APPS

Existing fault-tolerance techniques follow two main strategies: error processing and fault treatment. Error processing aims to remove errors from the application state while fault treatment aims to prevent the activation of faults in the application. Error processing is generally implemented by substituting an error-free state in place of an erroneous state or by compensating for the error with redundancy. In the case of fault treatment, the fault needs to be diagnosed in advance before an appropriate passivation technique can be applied. Depending on the nature of the faults, early diagnostics or state substitutions may not always be possible. Also, fault passivation may not always result in a behavior that is compliant with the specification the application. Future research should propose recovery mechanisms that take into account the dynamic nature of the cloud and ensure that the behavior of a cloud app remains compliant with its specification. Also, when implementing recovery patterns, it is important to create the recovery blocks at a granularity level that ensures the containment of the fault while minimizing the execution time and costs of replacement blocks. The identification of the right granularity of containment for a fault requires a good knowledge on the nature of the fault. Fault-tolerance design patterns such as those proposed by Hanmer [5] can be adapted to achieve this goal. An example of recovery mechanisms that aim to minimize recovery time and cost is the Recursive restartability [10], which consists in segregating components that often fail but recover quickly from components that fail infrequently but recover slowly; allowing the system to tolerate restarts at multiple levels. Recursive restartable designs restart an optimal number of components in case of a failure; allowing a system to recover quickly. Recently, authors like Zheng et al. [17] and Xu et al. [14] have proposed component ranking approaches that can be used to select components in cloud apps on which recovery mechanisms should be implemented. The components are ranked based on their characteristics, their invocation relationships, and their invocation frequencies.

4. CONCLUSION

In this position paper, we have surveyed some research work on the dependability of cloud apps and discussed the

[1] http://www.programmableweb.com/

implementation of fault-tolerance mechanisms in cloud apps. Complementary to recent works that have proposed approaches based on virtualization to improve the dependability of cloud apps, in this position paper, we argue for the implementation of recovery mechanisms in the architecture of cloud apps to make them robust to faults and hence highly dependable. We advocate for the use of fault-tolerance mechanisms to enable fault containment and recovery at the application level. The availability of multiple redundant services and resources in cloud environments offer the possibility to achieve this goal. However, because of the dynamic nature of the cloud and since existing fault-tolerance mechanisms assume that a fixed amount of hardware and system resources is bounded statically to the fault tolerant structures, it is challenging to include fault tolerance mechanisms in the architecture of a cloud apps. We suggest that future research works propose recovery mechanisms that take into account the dynamic nature of the cloud and preserve the compliance between the specification and the behavior of the cloud apps. These recovery mechanisms will reduce the recovery time of cloud apps and therefore improve their dependability and reduce the amount of money lost during service-downtime.

5. REFERENCES

[1] A. Avizienis. The methodology of n-version programming. *Software Fault Tolerance, M. R. Lyu (ed.)*, page 23âĂŞ46, 1995.

[2] A. Bondavalli, S. Chiaradonna, F. D. Giandomenico, and J. Xu. An adaptive approach to achieving hardware and software fault tolerance in a distributed computing environment. *Journal of Systems Architecture*, 47(9):763–781, 2002.

[3] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 161–174, Berkeley, CA, USA, 2008. USENIX Association.

[4] J. Edstrom and E. Tilevich. Reusable and extensible fault tolerance for restful applications. In G. Min, Y. Wu, L. C. Liu, X. Jin, S. A. Jarvis, and A. Y. Al-Dubai, editors, *TrustCom*, pages 737–744. IEEE Computer Society, 2012.

[5] R. Hanmer. *Patterns for Fault Tolerant Software*. Wiley, Chichester, England, 2007.

[6] A. Israel and D. Raz. Cost aware fault recovery in clouds. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 9–17, 2013.

[7] K. Kourai and S. Chiba. A fast rejuvenation technique for server consolidation with virtual machines. In *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on*, pages 245–255, 2007.

[8] M. R. Lyu. *Software Fault Tolerance*. Trends in Software, Wiley, 1995.

[9] Manitra. Fault tolerant web service framework. August 2008.

[10] D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft. Recovery oriented computing (roc): Motivation, definition, techniques,. Technical report, Berkeley, CA, USA, 2002.

[11] S.Shankland. Amazon suffers u.s. outage on friday. *Retrieved on December 15, 2013*.

[12] K. Trivedi, K. Vaidyanathan, and K. Goseva-Popstojanova. Modeling and analysis of software aging and rejuvenation. In *Simulation Symposium, 2000. (SS 2000) Proceedings. 33rd Annual*, pages 270–279, 2000.

[13] B. Winterford. Stress tests rain on amazon's cloud. *IT News, Retrieved on December 15, 2013*, August 2009.

[14] H. Xu, Y. Xie, D. Duan, L. Chen, and J. Wu. Ftcrank: Ranking components for building highly reliable cloud applications. In *PAKDD Workshops*, pages 522–532, 2013.

[15] X. Xu, Q. Lu, L. Zhu, Z. Li, S. Sakr, H. Wada, and I. Webber. Availability analysis for deployment of in-cloud applications. In *Proceedings of the 4th International ACM Sigsoft Symposium on Architecting Critical Systems*, ISARCS '13, pages 11–16, New York, NY, USA, 2013. ACM.

[16] Z. Zheng, T. Zhou, M. Lyu, and I. King. Component ranking for fault-tolerant cloud applications. *Services Computing, IEEE Transactions on*, 5(4):540–550, 2012.

[17] Z. Zheng, T. C. Zhou, M. Lyu, and I. King. Component ranking for fault-tolerant cloud applications. *IEEE Trans. Serv. Comput.*, 5(4):540–550, Jan. 2012.