HAL archives-ouvertes.fr

BFT-Dep: Automatic Deployment of Byzantine Fault-Tolerant Services in PaaS Cloud

Bijun Li, Rüdiger Kapitza

▶ To cite this version:

Bijun Li, Rüdiger Kapitza. BFT-Dep: Automatic Deployment of Byzantine Fault-Tolerant Services in PaaS Cloud. 16th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS), Jun 2016, Heraklion, Crete, Greece. pp.109-114, 10.1007/978-3-319-39577-7_9 . hal-01434804

HAL Id: hal-01434804
https://hal.inria.fr/hal-01434804

Submitted on 13 Jan 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License

BFT-Dep: Automatic Deployment of Byzantine Fault-Tolerant Services in PaaS Cloud

Bijun Li* and Rüdiger Kapitza

TU Braunschweig
{bli,kapitza}@ibr.cs.tu-bs.de

Abstract. Cloud computing has been a massive trend over the recent years and eased the deployment of scalable distributed applications. While initially renting out virtual machines was the predominant form of cloud computing, nowadays Platform as a Service (PaaS) solutions are emerging. The main advantages of the latter are a faster and easier application deployment as well as built-in support for horizontal scalability. However, when it comes to services with more demanding dependability requirements, currently provided deployment mechanisms quickly become insufficient, forcing cloud customers to fall back into manual, self-made strategies.
In this paper, we present BFT-Dep, a framework for deploying Byzantine fault-tolerant (BFT) services in a PaaS cloud automatically. BFT-Dep leverages the existing PaaS functionality to address specific deployment requirements and provides tailored support to set up and manage replicated services. It flexibly integrates BFT protocols as an independent service layer, thereby alleviating the complexity that the deployment of such systems entails. An initial prototype of BFT-Dep has been implemented on top of the open-source PaaS platform OpenShift and a first evaluation shows its practicability.

Keywords: Automatic Application Deployment; Platform as a Service (PaaS) cloud; Byzantine Fault Tolerance

1 Introduction

Cloud computing [22] offers a new form of resource provisioning, where services and applications are no longer hosted on local computing resources but on shared ones provided by remote infrastructures. Customers, i.e. software developers, are offered with services delivered over the Internet without having to manage and maintain the underlying hardware or system software that hosts these services.

Although delivering virtual machines is the most common form of cloud offering, Platform as a Service (PaaS) clouds [7,12,13,16], are getting increasingly popular. PaaS clouds aim at helping customers to quickly deploy and run their

* This research was supported by Siemens Rail Automation Graduate School (iRAGS). We would thank Torgen Hauschild, Marcel Kessler, Philipp Markiewka, Matthias Natho, Manuel Nieke, Mathias Rudnik for their contributions to this paper.

applications without considering infrastructure management tasks. Contrary to lower-level cloud services, non-functional properties like horizontal scalability are often part of the built-in features of these platforms. This makes PaaS clouds particularly appealing for more critical applications. However, if the applications depend on state, current PaaS solutions fall short as they lack deployment strategies for dependable stateful services. When services need to be replicated for high availability, the deployment is not a question of configuring independent service instances anymore, but of setting up and coordinating collaborating and highly interconnected replicas. This becomes even more complex, if the systems are supposed to tolerate not only crash faults but also Byzantine faults. For all that, cloud customers have to implement and maintain custom solutions, contradicting the primary goal of an automatic deployment in PaaS clouds.

This paper presents BFT-Dep, a novel application deployment framework that automatically sets up Byzantine fault-tolerant (BFT) [6] applications in PaaS clouds. Based on state-machine replication [18], BFT is able to tolerate crashes as well as arbitrary faults. By encapsulating BFT replicas into PaaS conform services, BFT-Dep simplifies the deployment of replicated applications without requiring changes to the cloud itself, and handles coordination among replicas automatically. An initial realization of BFT-Dep has been implemented as an extension of the popular open-source PaaS cloud OpenShift [16] and by using BFT-SMaRt [3,4] to enable replication.

Research works, which integrate Byzantine fault-tolerance protocols into cloud infrastructures to tolerate Byzantine failures have been proposed [2,8,11,19]. Also at middleware level, BFT has been combined with multi-tier web services to guarantee their heterogeneous reliability requirements [15]. Unlike them, our work is targeting the automatic deployment of reliable applications as an enhancement and extension of PaaS cloud functionality thereby making BFT services easier to deploy and manage.

The remainder of the paper is organized as follows: Section 2 explains the system design of BFT-Dep, Section 3 shows an implementation of BFT-Dep on a popular open-source PaaS cloud with interim evaluation results, and Section 4 concludes the paper and indicates future works.

2 System Design

An integral part of replicated Byzantine fault-tolerant services is the agreement protocol. It is responsible for coordinating the order in which requests must be executed by the stateful replicas to ensure the consistency of non-faulty ones. Consequently, to enable an easy deployment of BFT systems, the agreement protocol has to be integrated into the PaaS platform. BFT-Dep achieves this by introducing a *BFT agreement layer* that is separated from the application replicas (see Figure 1). The BFT agreement layer can be realized using a common BFT protocol implementation [1,6,14]. In principle this follows separation of agreement and execution stages [21] and the typical assumptions for Byzantine fault-tolerant systems are made. A minimum of $3f + 1$ replicas are required to

tolerate up to f Byzantine faults. Faulty replicas may crash or behave arbitrarily and maliciously. However, they are unable to break cryptographic techniques so that they cannot corrupt authenticated messages without being noticed. Note that enforcing fault tolerance of the cloud infrastructure itself is out of scope, as it is an orthogonal issue and has been addressed before [5].

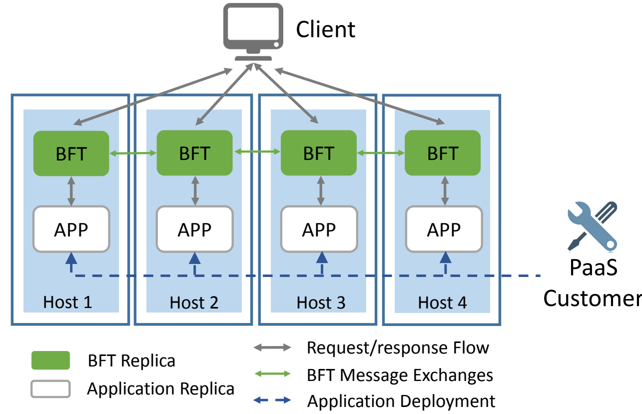

Fig. 1. Architecture of BFT-Dep

We build the BFT agreement layer by containerizing a BFT protocol, e.g. using Docker [9], into the software stack of a PaaS platform as a built-in service. Thus a BFT replica can be deployed and operated in the same way as any other service instances of a PaaS cloud. This way, BFT-Dep is able to leverage as much as possible the existing PaaS cloud facilities and is to a large extent immune to cloud architecture changes. When deploying applications, BFT-Dep first creates and configures a set of BFT replicas to establish the BFT agreement layer and then instantiates replicas for the customer's application. BFT-Dep guarantees that the application replicas together with their associated BFT replicas are distributed over different hosts of the cloud for fault independence. Each BFT replica exposes an access entry point to the clients. Upon receiving requests, the replicas of the BFT agreement layer first order them and then forward the requests to the corresponding application replicas. To do so, each BFT replica needs to connect to the application replica that locates on the same host to keep communication latency low. Application replicas then generate replies and send them back to the client through the BFT agreement layer, and the client will eventually verify the replies.

3 Implementation

We chose OpenShift Origin v3 [17] to implement BFT-Dep. Since BFT-Dep assumes a container-based PaaS model, it is not bound to a specific platform

but can be applied to most PaaS clouds built upon containers. Furthermore, BFT-SMaRt [3,4], a well-known BFT protocol implementation written in Java, was chosen for evaluation.

3.1 BFT Replicas Generation

BFT Image OpenShift uses Docker containers [9] that are built from Docker *images* for application deployment and management. A *Dockerfile* containing all necessary commands is needed for assembling an image. For building the BFT replica image, we use an off-the-shelf Java image as the basis and customize it to import the BFT-SMaRt source code. The necessary ports for connecting to other replicas as well as the associated application replica are declared. We eventually push the built image to a Docker Hub [10] repository, so that it can be conveniently imported to the PaaS cloud as an image stream for creating BFT containers.

BFT Pods and Services In OpenShift, the smallest deployable and manageable unit, the so-called *pod*, is responsible for holding the runtime environment of a set of containers. In BFT-Dep, we specify a JSON file to explicitly describe the features of BFT replica pods. A unique *name* as well as a *label* are assigned to distinguish pods. When creating replicated BFT pods, the BFT image is imported and the ports declared in the image need to be opened by the pod as well. BFT-Dep enforces those pods' allocations by assigning each one to an individual host of the PaaS installation, as replicas should always be distributed to different machines. We achieve this by specifying a *hostPort* item in the BFT pod description file so that according to the default scheduling policy, pods with the same hostPort number cannot be allocated onto the same host.

The unit *service* in OpenShift is defined as an abstraction of one or multiple pods, being responsible for exposing a stable service entry point regardless of underlying pod changes. A *label selector* is used to find all matching pods for the same service. We build BFT services upon individual BFT pods to forward external traffic. A temporary helper pod is used to configure and setup a cluster. It collects IP addresses of all BFT pods to update their host settings and then starts the BFT-SMaRt replica inside each pod. This way, BFT-SMaRt replicas can eventually connect to each other to agree on the order of client requests.

3.2 BFT Application Deployment and Networking

In OpenShift, customers use a *template* (a description file) to package application runtime dependencies for deployment. BFT-Dep offers a specific template for deploying BFT services and replicated applications. When using this template, BFT services are automatically generated and so are the application replicas, while their allocations are enforced by BFT-Dep as explained above. Each BFT service thereby connects to the application replica on the same host via socket connections. BFT services first order the received requests via public IPs, and

forward to the corresponding application replicas for execution, and eventually send the replies back to the clients.

3.3 Interim Evaluation

We installed OpenShift v3 on a cluster of AWS EC2 instances (all in US East), composed of one master and four plain nodes. Each EC2 instance is equipped with an elastic IP for public accesses from PaaS customers and clients.

We have conducted an interim evaluation of our preliminary BFT-Dep prototype using a simple key-value store application and the YCSB benchmark [20]. A workload of combined read and update operations is used for testing three deployment scenarios, as shown in Figure 2: 1) BFT and application deployed on EC2 instances together, 2) BFT and application deployed in OpenShift together, and 3) BFT and application deployed by BFT-Dep separately. Results of average latency indicate that the extra delay of using BFT-Dep introduced by the socket connections between BFT replicas and application services is low.

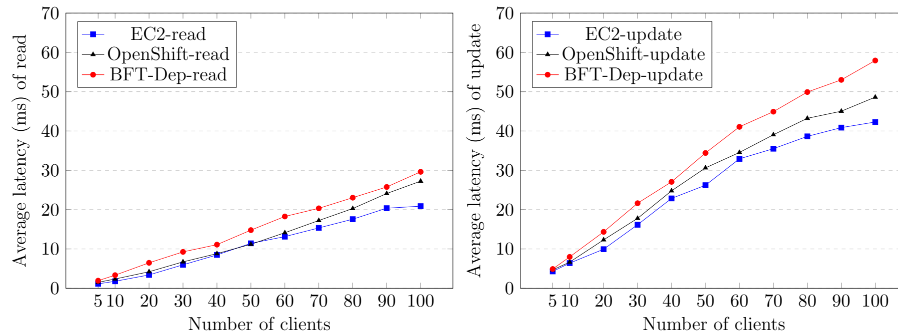

Fig. 2. Evaluation of read and update latencies with YCSB benchmark.

4 Conclusion and Future Works

We presented BFT-Dep, a framework leveraging existing PaaS facilities to address reliability demands of stateful application services. By integrating BFT protocols into the cloud platform as a built-in service layer, BFT-Dep offers customers the capability of automatically deploying and coordinating fault-tolerant applications while the PaaS platform itself can be left unchanged. Our preliminary BFT-Dep prototype has been implemented on top of OpenShift. The interim evaluation shows that the overhead that is added by a containerized BFT protocol implementation is low.

For future works, we plan to make BFT-Dep more generically applicable and enable reconfiguration and distribution changes as supported by BFT-SMaRt.

References

1. Behl, J., Distler, T., Kapitza, R.: Consensus-oriented parallelization: How to earn your first million. In: Proceedings of the 16th Annual Middleware Conference. pp. 173–184. Middleware '15, ACM (2015)
2. Bessani, A., Correia, M., Quaresma, B., André, F., Sousa, P.: Depsky: dependable and secure storage in a cloud-of-clouds. ACM Transactions on Storage (TOS) 9(4), 12 (2013)
3. Bessani, A., Sousa, J., Alchieri, E.E.: State machine replication for the masses with bft-smart. In: Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on. pp. 355–362. IEEE (2014)
4. Bft-smart. https://github.com/bft-smart/library
5. Brenner, S., Garbers, B., Kapitza, R.: Adaptive and scalable high availability for infrastructure clouds. In: Distributed Applications and Interoperable Systems. pp. 16–30. Springer (2014)
6. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proc. of the third USENIX Symp. on Operating Systems Design and Implementation (OSDI '99). pp. 173–186 (1999)
7. Cloud foundry. https://www.cloudfoundry.org/
8. Cogo, V.V., Nogueira, A., Sousa, J., Pasin, M., Reiser, H.P., Bessani, A.: Fitch: Supporting adaptive replicated services in the cloud. In: Distributed Applications and Interoperable Systems. pp. 15–28. Springer (2013)
9. Docker. https://www.docker.com/
10. Docker hub. https://hub.docker.com/
11. Garraghan, P., Townend, P., Xu, J.: Using byzantine fault-tolerance to improve dependability in federated cloud computing. International Journal of Software and Informatics 7(2), 221–237 (2013)
12. Google app engine. https://cloud.google.com/appengine/
13. Heroku. https://www.heroku.com/
14. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: speculative byzantine fault tolerance. ACM SIGOPS Operating Systems Review 41(6), 45–58 (2007)
15. Merideth, M.G., Iyengar, A., Mikalsen, T., Tai, S., Rouvellou, I., Narasimhan, P.: Thema: Byzantine-fault-tolerant middleware for web-service applications. In: Reliable Distributed Systems (SRDS), 2005 24th IEEE Symposium on. pp. 131–140. IEEE (2005)
16. Openshift. https://www.openshift.com/
17. Openshift origin v3. https://github.com/openshift/origin
18. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys 22, 299–319 (1990)
19. Verissimo, P., Bessani, A., Pasin, M.: The tclouds architecture: Open and resilient cloud-of-clouds computing. In: Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on. pp. 1–6. IEEE (2012)
20. Ycsb. https://github.com/brianfrankcooper/YCSB
21. Yin, J., Martin, J.P., Venkataramani, A., Alvisi, L., Dahlin, M.: Separating agreement from execution for byzantine fault tolerant services. In: Proc. of the nineteenth ACM SIGOPS Symp. on Operating Systems Principles (SOSP '03). pp. 253–267. ACM (2003)
22. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications 1(1), 7–18 (2010)