# Towards a Cloud Service for State-Machine Replication

Alexander Heß
Franz J. Hauck
alexander.hess@uni-ulm.de
franz.hauck@uni-ulm.de
Institute of Distributed Systems, Ulm University
Germany

## ABSTRACT

State-machine replication (SMR) is a well-known technique to achieve fault tolerance for services that require high availability and fast recovery times. While the concept of SMR has been extensively investigated, there are still missing building blocks to provide a generic offer, which automatically serves applications with SMR technology in the cloud. In this work, we introduce a cloud service architecture that enables automatic deployment of service applications based on customer-friendly service parameters, which are mapped onto an internal configuration that comprises the number of replicas, tolerable failures, and the consensus algorithm, amongst other aspects, The deployed service configuration is masked to large extent with the use of threshold signatures. As a consequence, a reconfiguration in the cloud deployment does not affect the client-side code. We conclude the paper by discussing open engineering questions that need to be addressed in order to provide a productive cloud offer.

## KEYWORDS

State-Machine Replication, Byzantine Fault Tolerance, Cloud Computing, Service Provisioning

## 1 INTRODUCTION

In recent years, major tech companies such as *Google* and *Facebook* have reported observations of so-called *soft errors* in their infrastructure [6, 14]. This term is used to describe arbitrary data corruption in memory cells, introduced by electromagnetic interferences or cosmic x-rays [12]. On the other hand, an increasing amount of public and critical infrastructure has been outsourced to the cloud for variety of reasons, which led to the creation of tailored cloud-service models [15]. While cloud-service providers rely on periodic backups to prevent data losses at larger scales, the recovery from a backup after a server failure can take multiple minutes up to hours. During recovery time the system will be unavailable, which could lead to severe consequences if the deployed service is crucial for work flows in the health care or public transportation domain.

For this reason, we aim to research building blocks that could be used to create a cloud-service model that guarantees fault-tolerant deployment of critical applications using State-Machine Replication. Hereby, we envision a *Framework-as-a-Service* cloud offer [17], where candidate applications have to adhere to the framework API, which provides an abstraction to the underlying fault-tolerance mechanisms. Similar to other cloud offers, required performance or availability figures of the service could be tweaked during the deployment process depicted in Figure 1, using customer-friendly parameters. The remainder of this paper is structured as follows: Section 2 provides a brief introduction to State-Machine Replication and Section 3 covers a selection of relevant previous work. In Section 4, we provide an overview of the internal architecture of our envisioned *SMRCloud* service, highlight the potential for parameterization of the proposed architecture, and finally discuss promising research ideas. Finally, we will briefly touch some open engineering problems in Section 5 and conclude in Section 6.

## 2 BACKGROUND

### 2.1 State-Machine Replication

State-Machine Replication (SMR) is a technique to achieve fault tolerance by deploying an application on a set of redundant servers [21]. It ensures two important service properties: *safety*, which means that clients are always able to obtain correct results, and *liveness*, which means that clients are actually able to retrieve service responses even in the presence of failures. This approach is based on three assumptions about the system: (i) the same shared initial application state throughout all replicas, (ii) a well-defined order imposed on the client requests, and finally (iii) deterministic execution of the received requests. While the first assumption is straightforward to realize in practice, ordering of client requests can be achieved with a variety of different approaches. Here, the most common approach is to use a consensus protocol [19],



*FGBS '23, März 06–07, 2023, Dresden, Germany*

https://doi.org/10.18420/fgbs2023f-02

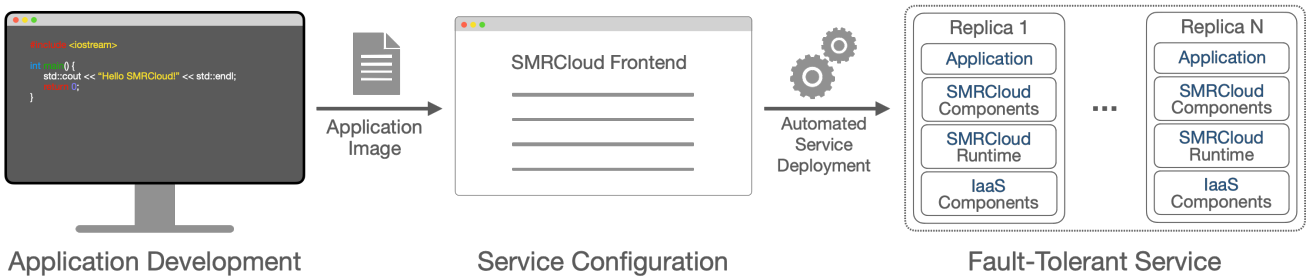


**Figure 1: The envisioned application deployment process with SMRCloud.**

which either provides a total ordering for all requests or a partial ordering for interfering requests, and requires collaboration among replicas. Finally, deterministic execution is required to prevent the replicas' application state from diverging. Here, the naive approach would be to simply use sequential execution, but this would drastically limit performance and would leave computational resources unused. A more sophisticated approach would be the use of a scheduler that enforces deterministic access on shared variables.

## 2.2 Failure Models

The most common failure models used in recent literature are the crash–fault-tolerance (CFT) and the Byzantine fault-tolerance (BFT) model. In an asynchronous environment, a CFT systems has to be composed of at least $2f + 1$ replicas, where up to $f$ replicas are allowed to crash within a given time window while the system still remains operational [18]. Typically clients issue their requests to all replicas of the system and accept the first result returned by any replica. In contrast, a BFT system requires at least $3f + 1$ replicas [4] in general, without relying on a hybrid fault-model. BFT systems typically incorporate more complex internal communication patterns as well as cryptographic functionality, which makes them less efficient than CFT systems. However, these systems can tolerate up to $f$ arbitrarily misbehaving nodes, while ensuring that clients are able to retrieve correct results by performing a quorum-based voting. The voting process requires that the client is aware of the correct quorum size $q$ for the system, which depends on the consensus protocol and the failure model, the number of replicas and the number of faults that can be tolerated.

# 3 RELATED WORK

The first practically-usable approaches for CFT [18] and BFT [4] SMR-Systems have already been proposed multiple decades ago. While the adoption of BFT SMR approaches in production systems is still relatively low due to the relatively large performance and resource overhead, there are quite a number of domain-specific approaches such as *Apache Zookeeper* [16] and *etcd* [5] which are already in use in productive systems. They build on SMR technology internally and can be used as a basis to build other highly-available services. However, since their SMR support is not application-agnostic they cannot be used to build generic applications. On the other hand, there are publicly available frameworks that can be used to build general-purpose BFT applications such as *BFT-SMaRt* [1] and *Concord-BFT* [10], although these frameworks require significant knowledge regarding the underlying fault-tolerance mechanisms in order to be used correctly. Further, none of these approaches provide functionality for automatic service deployment, which is also a crucial part for building a fault-tolerant production service. While orchestration tools like *Kubernetes* [3] could be used for the automatic deployment of the developed applications, their centralized node management and unsecured cryptographic key provisioning makes them unsuitable for BFT systems out of the box. Hence our goal is to bridge the functionality gap between these existing approaches and provide a straightforward way to deploy SMR systems in the cloud.

# 4 THE CLOUD SERVICE ARCHITECTURE

This section will provide insights into the proposed internal architecture of our envisioned cloud service *SMRCloud*. First, we will list a set of assumptions regarding the deployed application and the underlying components, then continue with a brief description of the individual components, and finally conclude with a set of promising research ideas that could advance our theoretical concept into a production system.

## 4.1 System Model

In our model, we assume that the deployed application is well-defined and uses the API provided by our framework as it is intended. This ensures that fault-tolerance mechanisms and the recovery procedure are able to ensure a consistent state. Further we assume, that the application does not contain security vulnerabilities or concurrency bugs, which could be used to bypass the fault-tolerant architecture by allowing an

attacker to compromise more than the tolerable number of machines. Finally, we assume that a future SMRCloud service provider is considered as a trusted party that relies on *Infrastructure-as-a-Service* (IaaS) components [17] for replica deployment. The reason for this assumption is the fact that the components that make up a fault-tolerant distributed system should only be affected individually in case of a hardware failure, an interrupted internet connection, or a power outage. As a consequence, the individual components should be deployed in different datacenters or availability zones to prevent a cascading effect of a single source of failure.

## 4.2 Replica Internals

The internal architecture and the communication channels between the individual components of a single SMRCloud replica are depicted in Figure 2. It can be observed that this multi-tier architecture includes a hardware layer, which is made up of IaaS components and serves as the basis for the runtime layer, composed of the replica's operating system, a container engine, and a replica management component. The use of a container engine allows us to create an abstraction layer from the underlying software and hardware stack in the uppermost layer, and as a consequence, a heterogenous set of replicas with similar performance numbers can be used to compose service instances. This approach increases service resilience against exploitable security vulnerabilities in individual software or hardware components. On the uppermost layer, the configurable SMRCloud components and the deployed application can be found. The deployed application is tightly coupled with a deterministic scheduler, and is connected to the SMRCloud components through a well-defined API. The individually displayed SMRCloud components have the following intended functionality:

The *Connection Handler* serves as the endpoint for the replica-to-replica communication, as well as the client communication where it serves as an entry point that masks the replica's internal configuration. While it is not feasible to hide the presence of the fault-tolerance mechanism without sacrificing *safety* and *liveness* guarantees of the system, it is possible to construct the communication handler such that configuration-agnostic client-side code can be utilized.

The *Cryptographic Module* is the major building block for achieving configuration-agnostic client communication. The idea is to use *BLS* threshold signatures [2] which enable a subset of replicas to aggregate their individual signatures into a single signature, which can be verified with the service's public key. This approach can not only eliminate the majority voting process on the client side, which is dependent on the consensus-related quorum sizes, but also reduce the number of signature verifications at client side.

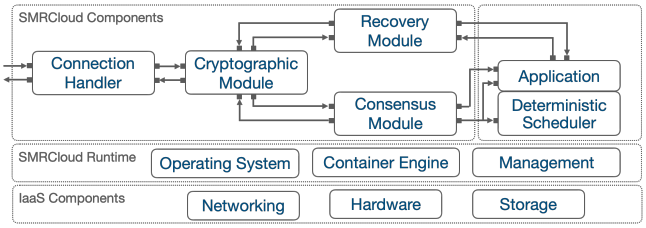


**Figure 2: The Envisioned Internal Architecture of a Single SMRCloud Replica.**

The *Consensus Module* could embed different consensus protocols which are based on the extensive preliminary work on optimization for specific use cases [8, 10]. This module could provide the largest space for parameterization, since there is no one-fits-all solution with outstanding performance in every possible use case [22].

The *Recovery Module* shall be responsible for the periodic creation of checkpoints, whereby the application state and consensus logs are stored on disk. While a naive approach would require to halt the replica's execution in order to create a consistent snapshot of the application state, recent research has shown that this process can be optimized based on certain assumptions [7]. This module also provides the option for parameterization, since the chosen checkpointing strategy provides a trade-off between the system's performance during normal operation and during the actual system recovery.

## 4.3 Research Questions

The above described SMRCloud architecture has a lot of potential for research ideas. One challenging aspect, which is typically omitted in recent publications, is the initial bootstrapping of the replica group and the distribution of the cryptographic keys. The cryptographic keys for the threshold signature scheme have to be generated with a distributed algorithm [9], because otherwise a single malicious instance could forge valid signatures. A rejuvenation or reconfiguration step of the system might require a regeneration of the cryptographic keys, if a certain number of replicas is removed or replaced. Although this might sound like we introduce an additional problem to the system, it actually enables us to hide the system reconfiguration from the clients to a large extent, as long as a majority of physical machines is kept during a single reconfiguration step.

Another interesting area of research is the design of the SMRCloud framework API, which is used for the communication between the application and consensus module, as well as the recovery module. Since the containerization approach would allow to deploy applications developed in any programming language, the SMRCloud API should also be

provided in a language-agnostic form. In this case, *gRPC* [11] could be a suitable candidate since it provides both high performance and code-generators for the most popular programming languages. Further, the deterministic scheduler might also be part of the SMRCloud framework API, since it has to be tightly integrated with the deployed application, and as a consequence it can not be part of the modular architecture. However, a configurable scheduler like *UDS* [13] could be used to provide further parameterization options.

## 5 FUTURE WORK

Solving all the research ideas that have been presented in this paper will not be sufficient for building a real-world SMR cloud service. Instead, the following open questions from the software and cloud engineering domain, need to be considered also. Firstly, in our system model we assume that the application code does not provide attack vectors against our fault-tolerant architecture. Although it is not feasible to erase all possible attack vectors in software, a *Fuzzing* framework could be leveraged to test the submitted application image in order to minimize the risk of a successful attack [20]. Further, a monitoring mechanism should be introduced in order to perform pro-active recovery or rejuvenation of individual machines to prevent an attacker from compromising a criticial amount of malicious nodes. Finally, a recovery procedure has to be designed that serves as a last resort to bridge the gap between theory and practice if the assumptions of the implemented fault models are violated.

## 6 CONCLUSION

In this work, we presented the potential of a symbiosis between state-machine replication, a well-established theoretical concept that can be leveraged to build fault-tolerant systems, and cloud computing, which is the state-of-the-art technology for service deployment. We outlined promising research aspects based on a conceptual cloud service model termed *SMRCloud*, and indicated some initial ideas that could lead towards the realization of such a model in the near future. Finally, we briefly discussed some remaining open questions which have to be solved in order to bridge the final gap between theoretical work and a production system.

## REFERENCES


[1] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. 2014. State Machine Replication for the Masses with BFT-SMART. In *44th Annual IEEE/IFIP Int. Conf. on Dep. Syst. and Netw. (DSN)*. IEEE Comp. Soc., 355–362. https://doi.org/10.1109/DSN.2014.43

[2] Dan Boneh, Ben Lynn, and Hovav Shacham. 2004. Short Signatures from the Weil Pairing. *J. Cryptol.* 17, 4 (2004), 297–319. https://doi.org/10.1007/s00145-004-0314-9

[3] Eric A. Brewer. 2015. Kubernetes and the path to cloud native. In *6th ACM Symp. on Cloud Comp. (SoCC)*. ACM, 167. https://doi.org/10.1145/2806777.2809955

[4] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* 20, 4 (2002), 398–461. https://doi.org/10.1145/571637.571640

[5] CoreOS. 2013. etcd. https://etcd.io/docs/

[6] Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, and Sriram Sankar. 2021. Silent Data Corruptions at Scale. *CoRR* abs/2102.11245 (2021).

[7] Michael Eischer, Markus Büttner, and Tobias Distler. 2019. Deterministic Fuzzy Checkpoints. In *38th Symp. on Rel. Distr. Sys. (SRDS)*. IEEE, 153–162. https://doi.org/10.1109/SRDS47363.2019.00026

[8] Michael Eischer and Tobias Distler. 2021. Egalitarian Byzantine Fault Tolerance. In *2021 IEEE 26th Pacific Rim Int. Symp. on Dep. Comp. (PRDC)*. 1–10. https://doi.org/10.1109/PRDC53464.2021.00019

[9] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *Advances in Cryptology – Int. Conf. on the Theory and Appl. of Crypt. Techn. (EUROCRYPT) (LNCS, Vol. 1592)*. Springer, 295–310. https://doi.org/10.1007/3-540-48910-X_21

[10] Guy Golan-Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: A Scalable and Decentralized Trust Infrastructure. In *49th Annual IEEE/IFIP Int. Conf. on Dep. Sys. and Netw. (DSN)*. IEEE, 568–580. https://doi.org/10.1109/DSN.2019.00063

[11] Google. 2016. gRPC – A high performance, open source universal RPC framework. https://grpc.io/docs/

[12] Qiang Guan, Nathan DeBardeleben, Sean Blanchard, and Song Fu. 2015. Empirical Studies of the Soft Error Susceptibility OfSorting Algorithms to Statistical Fault Injection. In *Proc. of the 5th Worksh. on Fault Tol. for HPC at EXtreme Scale (FTXS) (FTXS '15)*. ACM, 35–40. https://doi.org/10.1145/2751504.2751512

[13] Franz J. Hauck and Jörg Domaschka. 2016. UDS: A Unified Approach to Deterministic Multithreading. In *36th IEEE Int. Conf. on Distr. Comp. Sys. (ICDCS)*. IEEE, 755–756. https://doi.org/10.1109/ICDCS.2016.73

[14] Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E. Culler, and Amin Vahdat. 2021. Cores That Don't Count. In *Worksh. on Hot Topics in Oper. Sys. (HotOS)*. ACM, New York, USA, 9–16. https://doi.org/10.1145/3458336.3465297

[15] Wei Huang, Afshar Ganjali, Beom Heyn Kim, Sukwon Oh, and David Lie. 2015. The State of Public Infrastructure-as-a-Service Cloud Security. *ACM Comp. Surv.* 47, 4, Article 68 (jun 2015). https://doi.org/10.1145/2767181

[16] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. 2010. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX Ann. Techn. Conf. (ATC)*.

[17] Steffen Kächele, Christian Spann, Franz J. Hauck, and Jörg Domaschka. 2013. Beyond IaaS and PaaS: an extended cloud taxonomy for computation, storage and networking. In *IEEE/ACM 6th Int. Conf. on Utility and Cloud Comp. (UCC)*. 75–82. https://doi.org/10.1109/UCC.2013.28

[18] Leslie Lamport. 2002. Paxos Made Simple, Fast, and Byzantine. In *Proc. of the 6th Int. Conf. on Princ. of Distr. Sys. (OPODIS) (Studia Informatica Universalis, Vol. 3)*. Suger, Saint-Denis, rue Catulienne, France, 7–9.

[19] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (1982), 382–401. https://doi.org/10.1145/357172.357176

[20] Jun Li, Bodong Zhao, and Chao Zhang. 2018. Fuzzing: a survey. *Cybersecurity* 1, 1 (2018), 6. https://doi.org/10.1186/s42400-018-0002-y

[21] Fred B. Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comp. Surv.* 22, 4 (1990), 299–319. https://doi.org/10.1145/98163.98167

[22] Atul Singh, Tathagata Das, Petros Maniatis, Peter Druschel, and Timothy Roscoe. 2008. BFT Protocols Under Fire. In *5th USENIX Symp. on Netw. Sys. Des. & Impl. (NSDI)*. USENIX Assoc., 189–204.