Efficient Checkpointing in Byzantine Fault-Tolerant Systems

Michael Eischer Tobias Distler
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

ABSTRACT

Distributed Byzantine fault-tolerant systems require frequent checkpoints of the application state to perform periodic garbage collection and enable faulty replicas to recover efficiently. State-of-the-art checkpointing approaches for replicated systems either cause significant service disruption when the application state is large, or they are unable to produce checkpoints that are verifiable across replicas. To address these problems we developed and evaluated *deterministic fuzzy checkpointing*, a technique to create consistent and verifiable checkpoints in parallel with request execution.

KEYWORDS

Checkpointing, State-Machine Replication, Fault Tolerance

1 PROBLEM STATEMENT

Distributed Byzantine fault-tolerant (BFT) systems [1, 2] provide resilience against arbitrary hardware or software failures by processing a client request on multiple application replicas and voting over the determined results. To ensure that non-faulty replicas produce consistent and therefore comparable results, a BFT system relies on a fault-tolerant agreement protocol [2, 3, 4, 5] guaranteeing that all non-faulty replicas execute all requests in the same order. Participating in an agreement protocol requires replicas to repeatedly create, exchange, and store new protocol messages that may only be garbage collected after their effects manifested in a stable application-state checkpoint. To generate such a checkpoint, each replica periodically takes a snapshot of all application objects and verifies the snapshot's correctness by comparing it to the snapshots of other replicas. Besides garbage collection, checkpoints also play an important role in the recovery of faulty replicas [2, 6, 7, 8].

The state-of-the-art approach for BFT systems to ensure that a checkpoint comprises a consistent application-state snapshot is to temporarily suspend request processing while serializing all application objects into a dedicated buffer [2]. As a consequence, this technique usually decreases system availability, especially for applications with large states. Some in-memory databases from the domain of crash-tolerant systems circumvent this problem by implementing fuzzy checkpoints [9, 10, 11], that is, checkpoints that are created while the database processes transactions. Unfortunately, this approach cannot be directly applied to BFT systems as the fuzzy checkpoints provided by different non-faulty replicas are not guaranteed to represent the application state at the same point in time.

DOI: https://doi.org/10.18420/fbsys2019-01

2 APPROACH

To address the problems associated with existing approaches, we developed *deterministic fuzzy checkpointing* (DFC), a technique that enables BFT systems to produce consistent and comparable checkpoints without the need to suspend request processing. In the following, we summarize the most important aspects of DFC; for more details on the concept and its implementation please refer to [12].

General Concept. As shown in the figure, using DFC all replicas in the system create checkpoints for specific statically configured positions in the sequence of requests determined by the agreement protocol. In particular producing a checkpoint requires two phases: (1) During the *state-capture phase*, a replica starts a dedicated checkpointer thread that takes a fuzzy snapshot S by iterating over the entire application state and serializing all objects. In addition, the replica records a list M of all modifications to the application state that occur after the checkpointer thread started. With the checkpointer thread running in parallel with request execution, at the end of the state-capture phase the snapshot S does not necessarily comprise the latest version of each object. Furthermore, as each replica individually selects the starting point of state capture, the fuzzy snapshots produced by non-faulty replicas are likely to differ. (2) The subsequent *checkpoint-completion phase* therefore is responsible for making the snapshots consistent and comparable across replicas. For this purpose, each replica applies all modifications from list M to the fuzzy snapshot S, resulting in all objects in the snapshot to eventually be up to date at the end of the phase.

Implementation Variants. Having analyzed several BFT system architectures, we identified different ways for integrating DFC. On the one hand, it is possible to implement the checkpointing logic in a generic and application-agnostic manner, for example, as part of the replication middleware. On the other hand, the responsibility for object-modification tracking and capturing may be moved to the application, thereby enabling a more efficient implementation.

3 EVALUATION

We implemented DFC in the REFIT replication library [4, 5, 13, 14] and conducted experiments with a key-value store that manages its

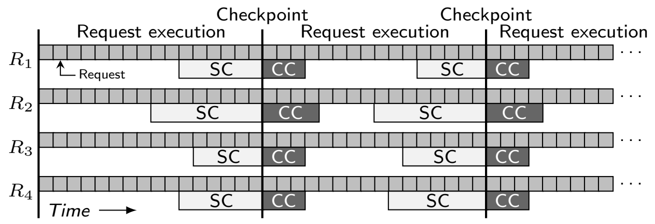

Figure: Using deterministic fuzzy checkpointing, all replicas (R_1,...,R_4) perform both state capture (SC) as well as checkpoint completion (CC) in parallel with request execution.

state in a SQLite database in memory. Our measurements show that for a 3 GB state (750,000 objects), despite all data residing in memory, serialization takes about 4.7 seconds. During this time, a replica is unable to process requests when using the traditional checkpointing approach. In contrast, relying on DFC it is possible to keep the service available throughout the entire checkpointing process.

4 CONCLUSION

The DFC technique enables BFT systems to efficiently create consistent application-state checkpoints in parallel with request execution and consequently leads to an increase in system availability.

Acknowledgments: This work was partially supported by the German Research Council (DFG) under grant no. DI 2097/1-2 (“REFIT”).

REFERENCES

[1] Fred B. Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22, 4, 299–319.

[2] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20, 4, 398–461.

[3] Alysson Bessani, João Sousa, and Eduardo E. P. Alchieri. 2014. State machine replication for the masses with BFT-SMaRt. In *Proceedings of the 44th International Conference on Dependable Systems and Networks (DSN ’14)*, 355–362.

[4] Tobias Distler, Christian Cachin, and Rüdiger Kapitza. 2016. Resource-efficient Byzantine fault tolerance. *IEEE Transactions on Computers*, 65, 9, 2807–2819.

[5] Michael Eischer and Tobias Distler. 2019. Scalable Byzantine fault-tolerant state-machine replication on heterogeneous servers. *Computing*, 101, 2, 97–118.

[6] Paulo Sousa, Alysson Neves Bessani, Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. 2007. Resilient intrusion tolerance through proactive and reactive recovery. In *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing (PRDC ’07)*, 373–380.

[7] Tobias Distler, Rüdiger Kapitza, and Hans P. Reiser. 2010. State transfer for hypervisor-based proactive recovery of heterogeneous replicated services. In *Proceedings of the 5th "Sicherheit, Schutz und Zuverlässigkeit" Conference (SICHERHEIT ’10)*, 61–72.

[8] Tobias Distler, Rüdiger Kapitza, Ivan Popov, Hans P. Reiser, and Wolfgang Schröder-Preikschat. 2011. SPARE: Replicas on hold. In *Proceedings of the 18th Network and Distributed System Security Symposium (NDSS ’11)*, 407–420.

[9] Robert B. Hagmann. 1986. A crash recovery scheme for a memory-resident database system. *IEEE Transactions on Computers*, 9, 839–843.

[10] Kenneth Salem and Hector Garcia-Molina. 1989. Checkpointing memory-resident databases. In *Proceedings of the 5th International Conference on Data Engineering (ICDE ’89)*, 452–462.

[11] Jun-Lin Lin and Margaret H. Dunham. 1996. Segmented fuzzy checkpointing for main memory databases. In *Proceedings of the 11th Symposium on Applied Computing (SAC ’96)*, 158–165.

[12] Michael Eischer, Markus Büttner, and Tobias Distler. 2019. Deterministic fuzzy checkpoints. In *Proceedings of the 38th Symposium on Reliable Distributed Systems (SRDS ’19).*

[13] Tobias Distler and Rüdiger Kapitza. 2011. Increasing performance in Byzantine fault-tolerant systems with on-demand replica consistency. In *Proceedings of the 6th European Conference on Computer Systems (EuroSys ’11)*, 91–105.

[14] Bijun Li, Wenbo Xu, Muhammad Zeeshan Abid, Tobias Distler, and Rüdiger Kapitza. 2016. SAREK: Optimistic parallel ordering in Byzantine fault tolerance. In *Proceedings of the 12th European Dependable Computing Conference (EDCC ’16)*, 77–88.