BFT-SMART — STATE MACHINE REPLICATION

Introduction

State Machine Replication (SMR) [1] is a classical fault-tolerance technique in which a set of service replicas can be consistently updated in such a way that the crash of a subset of some of them does not prevent the service to be provided. This technique has been extensively used to implement critical systems (e.g., datastores, coordination services like Zookeeper) in internet-scale infrastructures (e.g., Google, MSN, Yahoo!).

The last decade saw an impressive theoretical progress on Byzantine Fault-Tolerant (BFT) SMR, in which crashes, data corruptions and intrusions are tolerated. However, almost none of these techniques have been deployed in practical systems. One of the key reasons for this situation is the fact that there is no robust implementation of BFT SMR available, just proof-of-concept prototypes. This situation makes it difficult to use this technique, since implementing a BFT SMR protocol is far from trivial, with many subtleties that may lead even specialists to commit mistakes.

In TClouds we addressed the challenge of implementing a SMR library tolerating not only crashes, but also Byzantine faults. This library, called BFT-SMaRt, implements all distributed protocols required by SMR [2,3], targeting not only high-performance, but also completeness (implementing all corner cases) and extensibility.

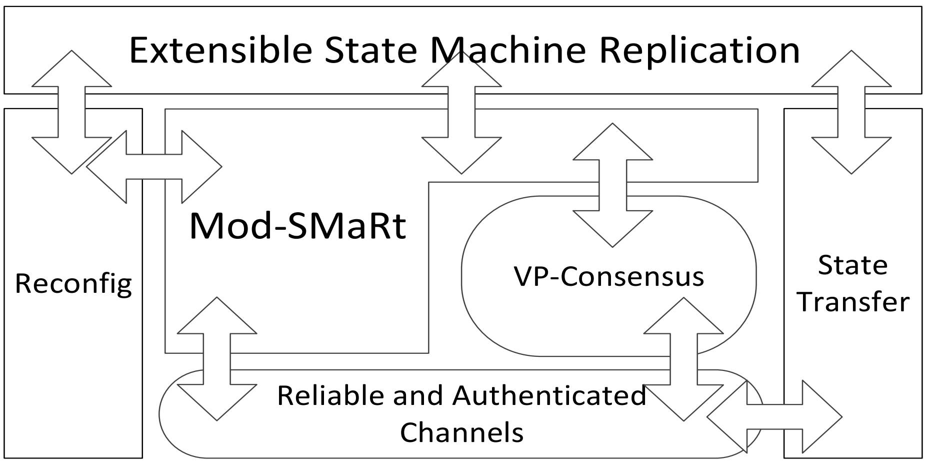
Extensible State Machine Replication
Mod-SMaRt
Reconfig
VP-Consensus
State Transfer
Reliable and Authenticated Channels

Figure 1: BFT-SMaRt Architecture.

BFT-SMaRt aims not only to bridge the gap of the absence of BFT SMR implementation, but also to provide a Java-based open-source implementation for state machine replication in general, since as far as we know, there is no crash fault-tolerant SMR framework available on the web. BFT-SMaRT can be used both to implement experimental next-generation dependable services and as a robust codebase for developing new protocols and replication techniques.

Key Features

Simplicity. Our emphasis on correctness and completeness made us avoid the use of fragile optimizations that could bring extra complexity or add unnecessary corner cases to the system. This emphasis also made us choose Java instead of C/C++ as the implementation language. Somewhat surprisingly, even with these design choices, the performance of our system is still better than that of some C-based SMR prototypes.

Modularity. BFT-SMaRt implements the Mod-SMaRt protocol [3], a modular SMR protocol that uses a well-defined consensus module in its core [2]. On the other hand, systems like PBFT [4] are implemented in a monolithic way, without a clear separation between protocols. In our opinion, modular alternatives tend to be easier to implement and reason about, when compared to monolithic protocols. Besides such basic protocols, BFT -SMaRt also implements state transfer and reconfiguration.

Reconfiguration and State Transfer. All previous BFT SMR systems assume a static system that cannot grow or shrink over time. BFT-SMaRt, on the other hand, considers a dynamic system model where replicas can join and leave the service group. This model adds the challenge of how to make the new replicas obtain the current state of the service in order to ensure consistency for executing the next operations issued to the system. We addressed this by devising a state transfer protocol that is triggered after a replica joins the group. The same protocol is also used for recovering crashed replicas (after a restart).

Extensible API. Our library encapsulates all the complexity of SMR inside a simple and extensible API that can be used by programmers to implement deterministic services. If the application requires advanced features not supported by this basic programming model, these features can be implemented with a set of plug-ins both at the client and at the server.

Multi-core awareness. BFT-SMaRt takes advantage of ubiquitous multicore server architectures to improve some costly processing tasks on the critical path of the protocol. In particular, our preliminary tests show that, when configured with client public-key signatures for added security a replica can process around 5K messages/second in single core servers, 20K messages/second in 4-core servers, 32K messages/sec in 8-core servers, 32K messages/sec in 8-core servers and up to 48K messages/sec in 16-core servers (all tests done with 1024 RSA and 5-byte requests.

High Performance. As mentioned before, BFT-SMaRt was built for correctness, modularity and completeness. Nonetheless, the overall performance of the system is much better than competing (less robust) prototypes found on the Internet. For example, in our tests for ordering small requests and an empty service that just send a small reply, our testes show that BFT-SMaRt offers a peak throughput almost 3x better (132 Kops/s vs. 49 Kops/s) than PBFT [4] (the baseline implementation for BFT SMR).

References

[1] F. B. Schneider. Implementing-fault-tolerant service using the state machine approach: A tutorial. ACM Computing Surveys. 1990.

[2] C. Cachin. Yet another visit to Paxos. Technical report, IBM Research Zurich. 2009.

[3] J. Sousa and A. Bessani. From Byzantine consensus to BFT state machine replication: A latency-optimal transformation. EDCC'12, 2012.

[4] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems. 2002.

Where To Find BFT-SMaRt?

http://code.google.com/p/bft-smart/

Further Information

Further information about BFT-SMART can be found under Deliverable „D2.2.1— Preliminary Architecture of Middleware for Adaptive Resilience“.

Disclaimer

The TClouds project has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement number ICT-257243.

TClouds at a glance

Project number:
257243

TClouds mission:
- Develop an advanced cloud infrastructure that delivers computing and storage with a new level of security, privacy, and resilience.
- Change the perceptions of cloud computing by demonstrating the prototype infrastructure in socially significant application areas.

Project start:
01.10.2010

Project duration:
3 years

Total costs:
EUR 10.536.129

EC contribution:
EUR 7.500.000

Consortium:
14 partners from 7 different countries.

Project Coordinator:
Dr. Klaus-Michael Koch
coordination@tclouds-project.eu

Technical Leader:
Dr. Christian Cachin
cca@zurich.ibm.com

Project website:
www.tclouds-project.eu