Study and Evaluation of BFT-SMaRt

Dilip Teja
School of Computing and Information
University of Pittsburgh
Pittsburgh, United States
dip67@pitt.edu

***Abstract*—BFT-SMaRt is a Byzantine Fault Tolerant (BFT) state machine replication protocol that ensures the consistency of replicated state machines in distributed systems. In this paper, The protocol has been extensively evaluated through simulation and experimental studies. The experimental evaluations have shown that BFT-SMaRt is efficient in practice, achieving high throughput and low latency while tolerating Byzantine faults.**

***Index Terms*—BFT, SMR, Byzantine**

I. Introduction

Last fifteen years have seen an impressive amount of work on the protocols for Byzantine fault-tolerant(BFT) state machine replication(SMR). BFT-SMaRt is an open-source Java-based library implementing robust BFT and bridging the gap between theoretical work and practical implementation. It targets high performance in the fault-free execution and correctness when faulty replicas exhibit arbitrary behavior. Study of such high performance system will helps us understand what is takes to build such a system. Evaluating BFT-SMaRt was carried out using series of experiments where each experiment alters the tunables of the system and validate the correctness of the throughput and latency of the system. BFT-SMaRt which claimed to be a high performance BFT engine was able to prove itself with the results of experimental evaluations. The results of the evaluations demonstrate that the system can be used in variety of applications, including Blockchain, distributed databases, and Cloud Computing.

II. Overview of BFT-SMaRt

A. Design Principles:

1) Tunable fault model: By default, BFT-SMaRt tolerates non-malicious Byzantine faults. Besides that, it also provides tolerance to malicious Byzantine faults and a simplified SMR protocol similar to Paxos.

2) Modularity: BFT-SMaRt implements a modular SMR protocol that uses a well defined consensus primitive in its core. This alternative tend to be easier to implement and reason about.

3) Simple and Extensible Application Programming Interface: The library encapsulates all the complexity of the BFT SMR inside a simple API that can be used by programmers to implement deterministic services. Clients can use a simple invoke(command) method to send commands to replicas that implement an execute(command) method to process the command.

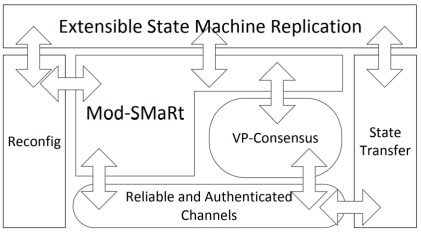

Fig. 1. BFT-SMaRt Modularity

4) Multi-core awareness: The library takes advantage of the multicore architecture of the servers to improve some costly processing tasks on the critical path of the protocol. The system throughput scales with the number of hardware threads supported by the replicas, especially when signatures are enabled and more computing power is needed.

B. Core Protocols:

1) Total Order Multicast : The total order multicast is achieved using Mod-SMaRt, a modular protocol which implements BFT SMR. During the Normal phase, clients send their requests to all the replicas and wait for their replicas. Total order is achieved through a sequence of consensus instances, each of them deciding a batch of client requests. When there is a fault, then Mod-SMaRt may switch to synchronization phase. During this phase, a new leader is elected and all replicas are forced to jump to the same consensus instance.

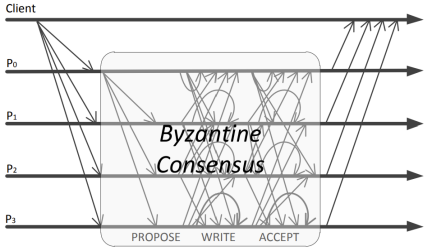

Fig. 2. BFT-SMART normal phase message pattern

2) State Transfer: In order to implement a practical SMR, BFT-SMaRt implements effective durability techniques like Log batches of operations in a single disk while these operations are being executed, Take snapshots at different points of the execution in different replicas to avoid stopping the system.

3) Reconfiguration: BFT-SMaRt provides an additional protocol that enables replicas to be added or removed from the system on-the-fly. Such processes can only be initiated by system administrators running a View manager client.

III. Evaluation

In this section, the results of the performance evaluation experiments are presented. The experiments include

- Micro-Benchmarks to evaluate the library's raw throughput and latency.
- Performance measurements of a BFT-SMaRt based service with fault scaling.
- Effect of increasing number of clients connected to the service

A. Experimental Setup:

All Experiments ran with three CFT and four BFT replicas hosted in the same machine. Clients and replicas are deployed in JRE Temurin-17.0.6+10 on Windows 11 . The machine has 16GB of memory and runs on 12th Gen intel(R) Core(TM) i7-12700H, 2300Mhz, 14 core(s) with hyper threading supporting 20 hardware threads. The processes communicate through the internal network.

B. Micro-Benchmarks:

This section reports the results of a set of micro-benchmarks used to evaluate the system.Such Benchmarks consist of an "empty" service implemented with BFT-SMaRt to perform raw throughput calculations at server side and latency measurements at the client side. Throughput measurements were gathered from the leader replica while the latency results are from one of the clients.

C. Experiment - 1 : Varying Request/reply size in BFT mode

As the system has a tunable fault model, for this experiment, we set the fault model to handle byzantine faults with f=1. In this setting, cluster need 4 servers in Byzantine mode which are kept running. A Benchmarking client is started with 10 clients, each client performing 10000 operations concurrently. This experiment is conducted with 4 different request/reply sizes of the operation. 0B/0B, 100B/100B, 1024B/1024B, 4096B/4096B. At the server side, after every 10000 operations processed, the latency and the throughput of the system are logged.

Fig 3 provides the throughput information of the system under different payloads. Interesting observation can be made that when there is an increase in the payload, there is an increase in throughput of the system which is opposite to what we generally expect. The probable reasoning for the behaviour is when the system's payload is increasing implies that the clients message size is increasing which force the BFT service to ramp up the usage of the host resources. This in turn increase the throughput due to high computational power. This behaviour is evident that BFT-SMaRt is multi-core aware and exploits the processing power of the host machine to achieve higher throughput. More details are provided in experiment-5.

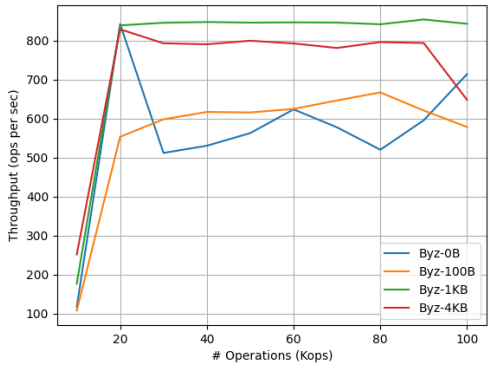

Fig. 3. Throughput of the system under different payloads in BFT mode

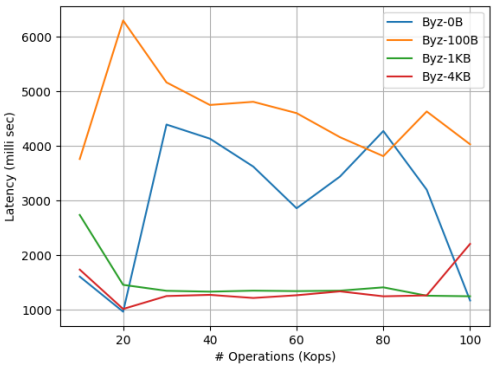

Fig. 4. Latency of the system under different payloads in BFT mode

Fig 4 provides the latency of the system under different payloads. From the plot, the operations with higher payload (4KB redline) have the lowest latency. This behavior can also be reasoned similar to one given to the throughput. When the system ramps up the host resources usage, the latency at which an operation is served is decreased which is indicative from the plot.

D. Experiment - 2 : Varying Request/Reply size in CFT mode

For this experiment, the fault model is now set to crash only mode (CFT) .This setting need 2f+1 servers to allow f faults. We perform the experiment with f=1. Same benchmarking client as Experiment-1 is started with 10 clients, each client performing 10000 operations concurrently. This experiment is conducted with 4 different request/reply sizes of the operation. 0B/0B, 100B/100B, 1024B/1024B, 4096B/4096B. At the server side, after every 10000 operations processed, the latency and the throughput of the system are logged.

From Fig 5, In CFT mode, with the increase in the payload size, the throughput is decreased as expected. When the payload increases, the system needs to process bigger sized requests which makes it take longer time, thereby reducing the throughput of the system.

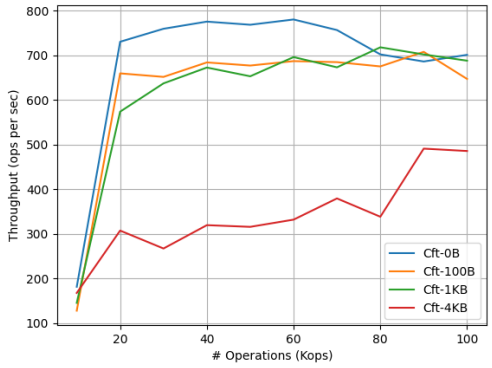

Fig. 5. Throughput of the system under different payloads in CFT mode

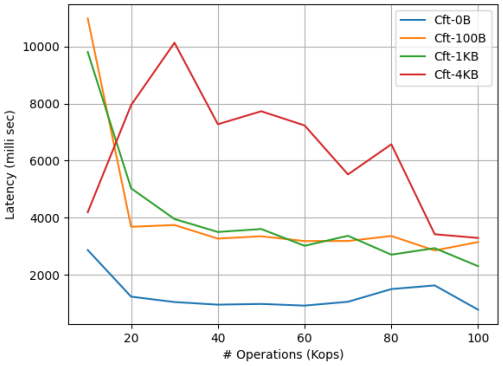

Fig. 6. Latency of the system under different payloads in CFT mode

In Fig 6, As the payload increases, the latency at which the operation is processed is also increased. This is expected in a CFT setting.
Overall, throughout and latency numbers in CFT mode displayed normal behaviour.

E. Experiment - 3 : BFT Vs CFT

This experiment compares the results of BFT and CFT settings of experiment 1 and 2 and make some inferences

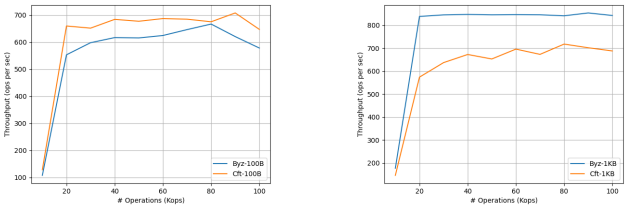

Fig. 7. Throughput of the BFT Vs CFT under different payloads

Fig 7 compares the throughput of the system in BFT and CFT settings. An interesting observation is throughput of the BFT is lower that CFT for smaller payload but surpasses it for higher payloads. Even though BFT requires more number of communication rounds when compared to CFT, given a fixed load, CFT has a higher throughput only for smaller payloads. But when it comes to the higher payloads, similar to what we observed in experiment-1, throughput of BFT dominates CFT. More details in Experiment-5.

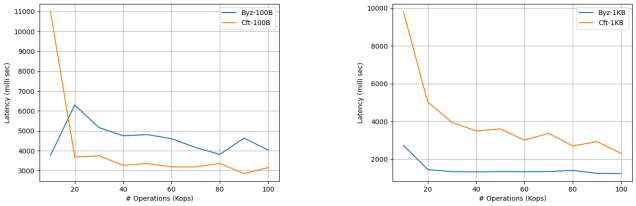

Fig. 8. Latency of the BFT Vs CFT under different payloads

Above figure compares the latency of the system in BFT and CFT settings. An interesting observation is latency of the BFT is higher that CFT for smaller payload and subceed CFT for higher payloads. The reasoning for this goes in the similar lines to that of throughput.

F. Experiment - 4: Fault scalability

In this experiment, we considered the impact of the number of replicas on the throughput of the system with different payloads. For all the configurations, the results show that performance of the BFT-SMaRt degrades with increase in f. This is because of the following reasons The protocol exchanges 2n(n-1) messages before achieving the consensus. So as f increases then the number of nodes n(2f+1 for CFT and 3f+1 for BFT) increases. With the increase of n, the number of messages that need to be exchanged also increases. Therefore the throughput decreases with f. We can also observe that with increase in request/reply size of the payload, the overall performance of the system is decreasing.

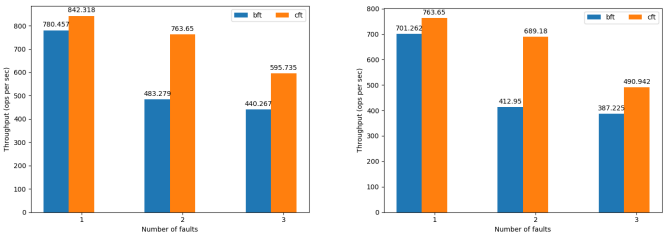

Fig. 9. Throughput of BFT-SMaRT for payload 0/0, 1024/1024 and f=1,2,3

G. Experiment - 5: Impact of Number of clients

In this experiment, we aim to evaluate the BFT-SMaRt throughput when we increase the number of clients connected. Number of connected clients are varied with a fixed request/reply size of 1024/1024 (10, 20, 30 clients). From Fig 10, With the increase in number of clients, the throughput of system increases. Typically, the throughput of a distributed system is expected to decrease as the number of clients increases, due to factors such as contention for resources

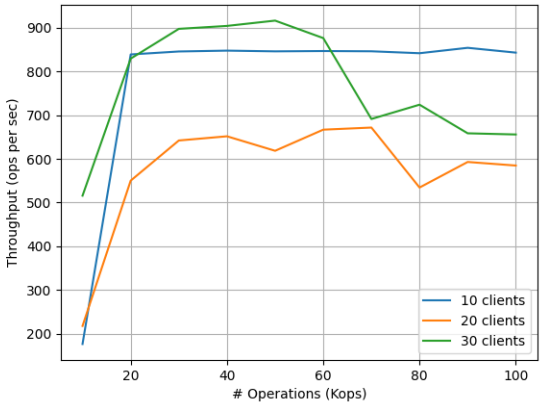

Fig. 10. Throughput of BFT-SMaRT for payload 1024/1024 with 10, 20, and 30 clients

and increased communication overhead. However, in the case of BFT-SMaRt, the throughput actually increases with an increasing number of clients. This is probably because it is designed to leverage the parallelism to improve performance. When the number of clients increases, system is able to process requests in parallel, by delegating the requests to multiple replicas for concurrent processing. This parallelism helps to reduce the overall processing time for a given set of requests, thereby increasing the throughput of the system. It's important to note that the performance of BFT-SMaRt may not continue to increase indefinitely as the number of clients increases, as there may be other factors that limit performance, such as network bandwidth or processing power. However, in general, BFT-SMaRt is designed to handle a large number of clients and maintain high throughput, even under high load conditions. Therefore, it is evident from the results of our evaluations that BFT-SMaRt is indeed a high performance system.

IV. Limitations and Future Work

Above sections presented few experiment results on the BFT-SMaRt. The possible limitations of the evaluation can be the following. The experiments are conducted on a single machine. Generally in a distributed setting. The experiments are conducted in different machines mimicking the real world scenario. This way the problem of resource contention is also reduces. As we conducted experiments in a single machine, The validation of the results can be tricky as the system is prone to hidden problems like resource contention of shared resources which might impact the throughput and latency. There can be more experiments that can be conducted which provides information about the client message signatures. As that can be well evaluated in a distributed setting , our work is limited and this experiments would be out of scope.
Potential future work would like to look at, will be to compare and evaluate other popular BFT engines such as Prime, PBFT, HotStuff with the present system. This could give a broader understanding of how well it is performing when compared to other protocols in certain conditions.

V. Conclusion

In conclusion, Our experiments show that the current implementation provides a very good throughput of medium-size messages.This evaluation show that the protocol is a promising solution for achieving strong consistency and high throughput. However, as with any distributed system, the performance and effectiveness of BFT-SMaRt will depend on the specific application and deployment environment, and further research and experimentation are needed to fully understand its strengths and limitations in different contexts.

References

[1] BFT-SMaRt: High-performance byzantine fault-tolerant state machine replication. http://code.google.com/p/bft-smart/
[2] Byzantine Fault-Tolerant (BFT) State Machine Replication (SMaRt) v1.2. https://github.com/bft-smart/library
[3] Sousa, J., Costa, P., Bessani, A., Correia, M., & Neves, N. F. (2018). State Machine Replication for the Masses with BFT-SMART. In Proceedings of the 48th International Conference on Dependable Systems and Networks (DSN'18) (pp. 211-222). IEEE.
[4] Sousa, J., Costa, P., Bessani, A., Correia, M. (2019). From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation. In Proceedings of the 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB) (pp. 1-8). IEEE.