Demo Abstract: ByzGame, a Visualized and Understandable BFT Consensus

James R. Clavin and Sisi Duan
{jclavin,sduan}@umbc.edu
University of Maryland, Baltimore County

Abstract

Byzantine Fault Tolerance (BFT) is the only generic technique that tolerates arbitrary failures in distributed systems, and can be used as a core primitive in building consensus in blockchains. Numerous BFT protocols have been proposed to improve scalability and throughput, but some have been found to have correctness issues resulting from system design and implementation. One of the major challenges is understand-ability of both the theory and implementation. Specifically, BFT consensus should be made understandable so that it is easy to reason about the correctness theoretically and also easy to understand the correctness of the implementation. To address the challenge, in August 2018 we began development of a web application called ByzGame. ByzGame is designed for visualizing BFT protocols, connecting the visualization with the backend BFT implementation, and making both BFT consensus theory and implementation more understandable.

ACM Reference Format:
James R. Clavin and Sisi Duan. 2019. Demo Abstract: ByzGame, a Visualized and Understandable BFT Consensus. In *20th International Middleware Conference Demos and Posters (Middleware Demos and Posters '19), December 9–13, 2019, Davis, CA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3366627.3368109

1 BFT Consensus

1.1 The Byzantine Generals Problem

The Byzantine Generals Problem is a classic problem in distributed systems that is not as easy to implement, adapt, and understand as it might seem to a systems architect [11]. A typical Byzantine Fault Tolerant (BFT) protocol assumes n servers which tolerates up to $f = \lfloor \frac{n-1}{3} \rfloor$ failures. A *quorum* of $\lceil \frac{n+f+1}{2} \rceil$ votes from different nodes are needed to reach consensus [18]. The first system to implement BFT was Practical Byzantine Fault Tolerance (PBFT) [4]. Several BFT protocols have since followed [2, 7–9, 13]. In the past few years, interest in BFT has increased due to its applicability in blockchains.

Middleware Demos and Posters '19, December 9–13, 2019, Davis, CA, USA
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7042-4/19/12.
https://doi.org/10.1145/3366627.3368109

1.2 Comprehensibility of BFT Consensus

Despite decades of research, BFT consensus still is not easily comprehended. The consensus theory alone is difficult to understand, much less whether an implementation matches the theory, as observed from multiple previous efforts [3, 5]. To the best of our knowledge, no BFT protocol has been created that has as a first class property being readily understood. We do, however, observe that the crash fault tolerant (CFT) protocol Raft [14] was designed to be more easily understood than Paxos, the existing state of the art in crash fault tolerance [10]. Just as Raft uses a visual and interactive game to teach the protocol, so too does our ByzGame. But, comparatively, we go a step further in ByzGame by connecting the visualization with a BFT backend codebase.

2 ByzGame: The Byzantine Generals Game

We have built ByzGame, a web-based interface that connects the web front end with a BFT implementation, so that users can directly manage the BFT nodes and visualize the message flow. Through such a process, users can do real world tests of the theory of BFT consensus, examine the BFT implementation, and identify issues with the protocol.

2.1 BFT-SMaRt

We utilize BFT-SMaRt, an open source java-based BFT library [16]. It has an API for re-use in other applications by either clients or servers. Clients can send messages using total ordered or unordered multicast, either synchronously or asynchronously [15]. The API provides the ability for an application to execute client messages, perform state transfer, and define what message to send in reply [15].

2.2 The Python shim

The ByzGame uses a Python shim to interact with BFT-SMaRt through both network sockets and using JPype [12]. The shim includes a sender thread that relays a message through the socket to a receiving replica thread. The receiving thread handles the request and sends back a message indicating that the request was processed. Figure 1 is a diagram of the communication process for the ByzGame, with 4 servers (referred to as replicas) running BFT-SMaRt, each with a Python shim.

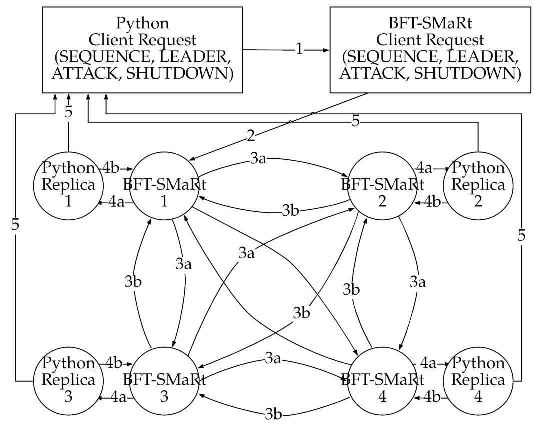
Python Client Request (SEQUENCE, LEADER, ATTACK, SHUTDOWN)
BFT-SMaRt Client Request (SEQUENCE, LEADER, ATTACK, SHUTDOWN)
1
2
5
Python Replica 1
BFT-SMaRt 1
Python Replica 2
BFT-SMaRt 2
Python Replica 3
BFT-SMaRt 3
Python Replica 4
BFT-SMaRt 4
3a
3b
4a
4b

Figure 1. ByzGame Design: Python + BFT-SMaRt.

2.3 Interface

The ByzGame is a Python web application that presents two system configuration inputs to users: n, the number of nodes - or "generals"; and the "target" - or the city against which the "generals" launch their attack. Upon selection of n, the system state variables for faults allowed, or f, and quorum size, or Q, are calculated and shown to the user. $\forall$ *replica* i, $1 \leq i \leq 38$, a node is placed on a global map. (38 is the max n the system supports; or, 39 cities total, one of which is the target). The user can then issue these commands.

- **START** initializes all servers in the system.
- **STOP** stops all servers in the system.
- **LAUNCH** gives the order to "Launch Attack".
- **RESET** puts the system state to the default setting where $n = 4$ and $f = 1$.

"Launch" sends a client request to the nodes running BFT-SMaRt, who return either a "1" indicating they will "attack," or a "0" otherwise. If the nodes reach consensus, the interface animates either an "attack" against, or "retreat" from, the target city, as in the original Byzantine generals game proposed in the 80s [11]. System logs at the nodes are accessible to illustrate message flow in the protocol. When failures occur, a user can search the logs to identify the cause.

3 Implementation

The ByzGame is hosted on an AWS EC2 t2.micro instance. Users from our university log in to ByzGame.com by authenticating using their university-provisioned Gmail account.

Excluding the BFT-SMaRt library, the ByzGame project has about 5,000 lines of code written in Python, JavaScript, and HTML. The web framework is the Python library Tornado [1]; it manages both the presentation layer and the network socket communication. The main Tornado server interacts with static HTML and JavaScript libraries. The global map uses geocoded JSON data presented through the JavaScript library D3. We use LevelDB [6] as the database for logging. Each replica stores its own log data and records client requests.

4 ByzGame Status

We believe ByzGame is an ideal tool for education because it gives students the means to configure and simulate a distributed system that is running a open source BFT protocol [17]. The map is intended to help visualize the locations of the servers, and underscore the distributed nature of BFT. Feedback on quorum size and allowable faults is dynamic and is designed to elicit realizations. E.g., certain n settings have the same Q value, so the number of servers in a system should be given careful consideration so as to not waste resources. Furthermore, students can better grasp BFT theory by not only examining the output of BFT-SMaRt, but the output they have caused. We are currently testing the ByzGame interface in a distributed systems course at the university. The game will supplement the lecture on consensus and BFT. An assignment will be given with the expectation that ByzGame provides the answers and a survey will gather feedback on user experience. Class performance on exam questions related to consensus and BFT will be compared to prior years' classes performance on similar questions.

References

[1] 2019. Tornado Web Server. https://www.tornadoweb.org/en/stable/. (2019).
[2] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology* 18, 3 (2005), 219–246.
[3] Christian Cachin and Marko Vukolić. 2017. Blockchain consensus protocols in the wild. In *DISC*. 1:1–1:16.
[4] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine fault tolerance and proactive recovery. *TOCS* 20, 4 (2002), 398–461.
[5] Tushar D Chandra, Robert Griesemer, and Joshua Redstone. 2007. Paxos made live: an engineering perspective. In *PODC*. ACM.
[6] Jeff Dean and Sanjay Ghemawat. 2012. LevelDB. *Retrieved* 1 (2012).
[7] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT Made Practical. In *CCS*. ACM, 2028–2041.
[8] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2018. SBFT: a scalable decentralized trust infrastructure for blockchains. *arXiv preprint arXiv:1804.01626* (2018).
[9] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2010. The next 700 BFT protocols. In *Eurosys*. ACM, 363–376.
[10] Leslie Lamport. 1998. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)* 16, 2 (1998), 133–169.
[11] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine generals problem. *ACM TOPLAS* 4, 3 (1982), 382–401.
[12] S Ménard and L Nell. 2006. JPype. (2006).
[13] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *CCS*. ACM, 31–42.
[14] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *ATC*. 305–319.
[15] snakejerusalem. 2016. BFT-SMaRt Programming Interfaces. (2016).
[16] João Sousa, Eduardo Alchieri, and Alysson Bessani. 2014. State machine replication for the masses with BFT-SMaRt. In *DSN*. 355–362.
[17] Joao Sousa, Alysson Bessani, and Marko Vukolić. 2018. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *DSN*. 51–58.
[18] Andrew S Tanenbaum and Maarten Van Steen. 2007. *Distributed systems: principles and paradigms*. Prentice-Hall.