Replicated and Consistent Distributed Data Storage

Roy Shadmon
CSE Department
University of California Santa Cruz
Santa Cruz, California

Niharika Srivastav
CSE Department
University of California Santa Cruz
Santa Cruz, California

***Abstract*—This project develops a verifiable mechanism for independent nodes receiving transmitted data to respond to client read-only queries. In addition, this system ensures that when a client sends a request, all the nodes process the request on the most recently processed state of the data. This mechanism also allows a client to determine which node(s) are not abiding by the service level agreement (SLA) if one node is significantly behind in committing data compared to the other nodes. This system manages crash faults. For example, if a node crashes, it can verifiably receive the data it failed to receive when it became unavailable as soon as it comes back up. The protocol used to implement this is the gossip protocol. Also, each node keeps a log containing all of its commits, which allow for a quick way to catch up a benign node.**

***Index Terms*—Distributed Database, P2P Network, Replicated storage, IoT, Fault Tolerance**

I. Introduction

In the world of big data, data is continually being recorded by some device and transmitted to some server to process and store the data efficiently. A client, also known as the data owner, is then able to request the server to retrieve the data transmitted from the device. In most cases today, data owners store their data using a single cloud provider such as Google Cloud, Microsoft Azure, or Amazon Web Services. If a client needs their data to be replicated, the data is replicated on multiple servers hosted by the single cloud provider. The problem with the common approach is that there is still no availability guarantee if data is hosted on one cloud [1,3]. What happens, for example, if there is an outage amongst Google Cloud servers [2]? Data is not truly replicated in an isolated manner if the same cloud provider hosts it, as the failure of one cloud provider causes a failure in the replication of the entire system. Yes, Google can host our data over multiple data regions. However, this relies upon requests to travel farther distances; hence, decreasing performance. If data were to be stored over multiple cloud providers, on the other hand, then there would be stronger availability guarantees. However, using multiple large cloud providers is rather expensive, and the setup of the system is not obvious. Besides, when a client sends a request to a server that is hosted within a data center, the request must travel through the data center to find the correct server, process the request, and then traverse through the data center again for the message to be sent back to the client [3]. This arduous request processing hinders the performance of a client's request, which can be avoided using new technologies. Using direct nodes (possibly nodes in P2P networks), we can leverage nodes that offer data storage and processing services by providing the client and devices direct access to making requests; these node servers can process requests on the edge of the network, which mitigates the need of a request and response traversing through a data center. In addition, we can replicate the data over multiple independent nodes that offer similar SLAs. For example, if a client wants to transmit their IoT data to nodes offering data services, we can set the data to be transmitted to all the nodes where we want the data to be replicated.

Assuming that the devices transmitting data is trusted meaning a device will always send the data to all nodes (since one cannot trust the data sent from an untrusted device), a requirement for this system is to ensure that when a client makes a request to the nodes, the nodes are all processing the request on consistent data. This is important because a client needs their data to be highly available in the case a node becomes benign (the node intends to act honestly, but fails for a short, finite period of time). In the first phase of this project, we assumed all nodes are honest, as well as; no node will crash. In the second phase, assumed all but one node can become benign for some finite period of time. For the implementation, we have not considered byzantine nodes. However, future work will definitely address a byzantine setting. In addition, we assume that all queries made by a client are exclusively read-only queries since updates and writes by clients are not needed in an IoT architecture. In this project we have developed a verifiable mechanism for independent nodes receiving transmitted data. The system ensures that when a client sends a request, all the nodes will process the request on the most recent processed state of the data. One possible method to do this is to develop an efficient batching mechanism where nodes agree to commit a predefined count of data. For example, nodes can agree to commit every 10 pieces of data received. This mechanism will also allow a client to determine which node(s) are not abiding by the SLA if one node is significantly behind in committing data compared to the other nodes.

After getting this mechanism to work and assuming nodes may not be benign, we attempted to support node recovery. For example, if a node crashes, it can verifiably receive the data it failed to receive when it becomes unavailable, using the gossip protocol. A gossip protocol[5] is an algorithm to implement peer-to-peer communication that is based on the

way epidemics spread. Distributed systems use P2P gossip to ensure that data is disseminated to all members of a group.
In addition, each node keeps a log containing all of its commits, which allow for a quick way to catch up with a benign node.

II. Description of the prototype

The system consists of a device, a client and multiple servers. The device continuously sends data to the servers where it's saved in a database. The client can currently only query for the value at a specific date (just like a key-value store) from any of the servers; the client must specify which server it wants to query. The servers, the client, and the device use PubNub[6], a global data stream network that acts as our node-to-node communication tool. PubNub is a real time Infrastructure as a Service (IaaS) tool that allows for decentralized entities and to send and receive messages through channels. A channel can be thought of as a host and port entry point where nodes can communicate with each other in real-time. In addition, every server maintains its own database instance and a server log where it keeps track of every key-value it commits to the database.
The overall architecture of our system is a single device that transmits data to multiple servers. Currently, a single client is then able to query a specific server regarding a key and the server will return a value. Our prototype also supports fault-tolerance, which is discussed in the following section.

III. Fault Tolerance

If a server were to fail, the server upon come up will communicate will active servers listening to the recovery channel. The node who needs to recover missed data will use its log to see what data has been committed. The steps to recover missed data is as such: the failed server looks at its first log item and last log item. It then sends this information to all other servers and those servers run a query to receive all the data before the log item and all the data after the last log item. Once the servers compute the result, they send the result to the initial requesting node.

IV. Description of the suggested byzantine fault-tolerance enhancement

To support byzantine fault-tolerance, we can simply use paxos for the servers to come to agreement on the contents of the data. In addition, we can skip all the phases and have each server send the client its result and the client would be the one determining which value is correct. Of course, this assumes that the client is trusted, and we will discuss this more why this is a valid assumption. This is something we are also still thinking about.

References

[1] Majadi, Nazia. (2012). Cloud Computing: Research Issues and Challenges.

[2] Marcos K. Aquilera, Brian Cooper, Yanlei Diao. Why does the Cloud Stop Computing?: Lessons from Hundreds of Service Outages (2016).

[3] M. Malathi, ”Cloud computing concepts,” 2011 3rd International Conference on Electronics Computer Technology, Kanyakumari, 2011, pp. 236-239. doi: 10.1109/ICECTECH.2011.5942089.

[4] T. Neudecker, P. Andelfinger and H. Hartenstein, ”Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network,” 2016 Intl IEEE Conferences on Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, 2016, pp. 358-367.

[5] K. Jenkins, K. Hopkinson and K. Birman, ”A gossip protocol for subgroup multicast,” Proceedings 21st International Conference on Distributed Computing Systems Workshops, Mesa, AZ, USA, 2001, pp. 25-30. doi: 10.1109/CDCS.2001.918682

[6] The Publish-Process-Subscribe Paradigm for the Internet of Things Bhaskar Krishnamachari, Kwame Wright July 2017