Tribune: An Externally Consistent Database Common Access API for the Global Data Plane

Matt Weber, Shiyun Huang and Lawrence Supian
Department of Electrical Engineering and Computer Sciences (EECS)
University of California, Berkeley
Email: matt.weber, jane.huang, lsupian@berkeley.edu

Abstract

The Global Data Plane (GDP) is a distributed data storage system with the objective of providing persistent data storage to devices in the internet of things. The GDP has a variety of useful properties including data security, a flat namespace, and location independent routing, that improve the availability and efficiency of universal data storage. As a key component in the Terraswarm project and the SwarmOS, the GDP provides log based storage and routing between sensors and actuators embedded in the physical world. Logs are the fundamental primitive of the GDP, but some CAAPIs (a Common Access Application Programming Interface) require stronger semantics. In this paper we introduce Tribune, a multi-writer log CAAPI that enforces Google Spanner-like external consistency. We compare the trusted Paxos based replication scheme with attack tolerant Byzantine Agreement and evaluate optimistic concurrency against the lock table-based concurrency control scheme used by Google Spanner. In a performance evaluation we found the optimistic algorithm to be generally faster than the comparable lock table approach, particularly when paired with Byzantine agreement. Our long term goal is to provide PTIDES style deterministic execution semantics for swarmlets in the SwarmOS that choose to access time-aware multi-writer logs.

1 Introduction

The Internet of Things (IoT) presents a new paradigm for computer systems, enabling technologies like context-aware apps, extensible cyber-physical systems, large scale sensor data collection for machine learning, and smart cities. But systems of the future will not come for free: engineers will need ubiquitous computing infrastructure in the form of platforms, services, and tools as a foundation for their work. The Terraswarm project [9] seeks to address the gap between the development resources of today and those needed to engineer the coming swarm of interconnected devices [12]. There are many dimensions of the problem to be considered, including verification tools, low-power sensors, programming models, and persistent universally available data storage. The last point in particular is of primary concern to swarmlet (an application in the swarm) developers who need data accessible across different usage modes and application contexts [4].

The Global Data Plane (GDP) [5] is a TerraSwarm project that seeks to extend upon the capabilities of the Cloud to meet the needs of decentralized and interoperable swarmlets. Oceanstore [14] provides a starting point, as data must be globally available, durable, private, secure, and efficiently accessible. Additionally, the GDP must support storage and distribution of streaming sensor data, which it achieves through log-based data storage in a flat address space. As such, the log is a principle component of the GDP, and could be used as the fundamental building block for arbitrarily complicated systems for information representation. This design choice does not limit the efficiency or expressiveness of the GDP, because the GDP supports a Common Access Application Programming Interface (CAPPI) for data representations with sophisticated semantics or complex structure. Although possible, the process of reconstructing a full database or key-value store from log data would be prohibitively expensive if every time the sophisticated data structure were needed it had to be built from logs and then immediately thrown away. A CAAPI can maintain the current state of the enhanced structure directly and function as a sort of GDP cache for a particular data representation.

Although CAAPIs can be used in the GDP to store data efficiently, they also have the capacity to enforce semantics on the behavior of operations on data. This work, Tribune, is an example of a CAAPI that enforces external consistency along with standard ACID semantics on the behavior of a multi-writer log. The general concept of this style of CAAPI is illustrated in Figure 1. Tribune's name is a reference to the role of an ancient Roman Tribune, an elected official with the power to intervene on behalf of the common people by vetoing legislation from the senate. In a similar respect, Tribune can control transactions that attempt to write to a protected log in this multi-writer merge style, and abort transactions that violate its semantics.

This paper is organized as follows: Section II provides background information on the database behavior we seek to emulate in Tribune. Section III gives an overview of the system architecture and design. Section IV goes into the implementation details for our most interesting algorithms. Section V elaborates on our test environment setup and presents benchmark results. In Section VI we address the direction of future work. We conclude with Section VII.

2 Background

Tribune's primary role is to enforce database semantics on a multi-writer log. We chose the semantics of Google Spanner's read write transactions [3] because Spanner provides a time-based external consistency guarantee and works at a global scale. External consistency establishes the invariant that if transaction A commits before transaction B begins as observed from the outside world, timestamp associated with transaction B in the database must be later than than A's timestamp. Spanner achieves this invariant through a combination of two phase locking within a Paxos group, time-aware two-phase commit between Paxos groups, and precise implementation of the TrueTime API as shown in Figure 2. TrueTime establishes an ordering between timestamps that reflects error in clock synchronization, i.e. it is unknown whether a timestamp proceeds another unless the difference in the timestamps exceeds the known upper bound on the offset between clocks. Spanner's high level architecture is shown in Figure 3. All of the tablets that are associated with a particular data model run on a set of replicas in the same Paxos group. The primary responsibility of the Paxos leader is to replicate the state of writes that go through it across the group. The leader also acts as a bottleneck for all read write transactions, which allows it to maintain a lock table that reflects the status of transactions currently in flight. Clients that want to perform read write transactions must acquire locks through two phase commit, and deadlocks are prevented with wound wait [16]. When a read write transaction needs to commit across paxos groups (a ”distributed transaction”), one paxos leader steps up for the role of participant leader and runs a transaction manager to organize the two-phase commit. Spanner also supports read only transactions and read transactions (the two are actually distinct types in Spanner) that can be performed at any replica without going through the paxos leader.

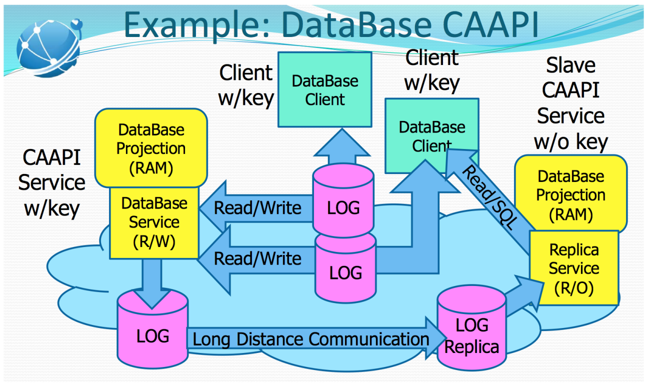

Figure 1. An illustration of a database CAAPI. Image from slide 17 of [5]

Method	Returns
$TT.now()$	$TTinterval$: $[earliest, latest]$
$TT.after(t)$	true if t has definitely passed
$TT.before(t)$	true if t has definitely not arrived

Figure 2. Spanner's TrueTime interface. Image from Table 1 of [3]. Note that t is a timestamp

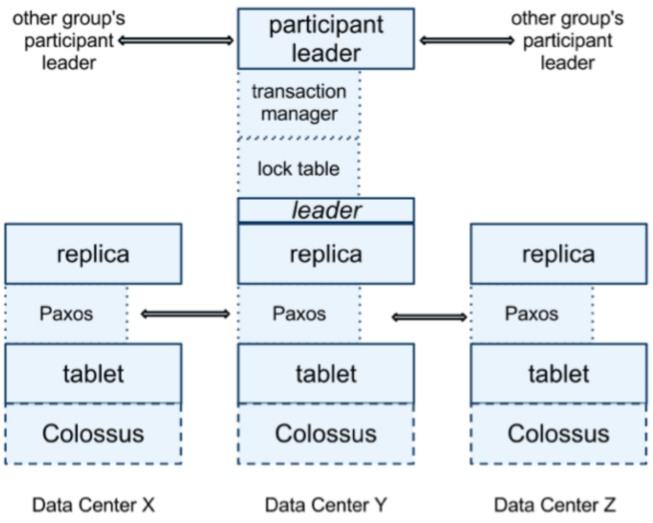

Figure 3. Spanner's Architecture. Image from Figure 2 of [3]

2.1 Time Synchronization

Spanner uses a complex global network of atomic and gps clocks to achieve time synchronization. An advantage of establishing very accurate clocks with very low drift is that very little communication is needed to preserve synchrony. IEEE 1588, the Precision Time Protocol [13] presents an alternative, ip based mechanism to achieve synchronization on the order of microseconds across a wired local area network. Truetime is used in spanner to determine when to pick timestamps for transactions in two-phase commit and how long to hold locks after committing within a paxos group. Note that without any kind of synchronization between clients, relativity makes it impossible to evaluate external consistency. The definition of external consistency requires that transactions outside the system exist on a shared timeline (because they must be comparable) which implies some way to compare times at two locations, whether quantitative or logical [6].

2.2 Consensus Algorithms and Durability

Consensus algorithms like Paxos [7] and Byzantine Agreement [8] **COMPLETE THIS SECTION**

Reed-Solomon encoding [1] is a good approach for long-term data storage when the top objective is reducing the probability that data will be lost. But for short term data storage, using a consensus algorithm to achieve agreement on the state of a set of replicas is a much faster approach. Perhaps the Reed-Solomon encoding data might be verified by a Byzantine Agreement ring as in [14] but in the short term, other methods become much more practical. As a long-term objective for the GDP, the system ought to achieve a balance between durable and responsive storage.

Another dimension of durability concerns the transfer of data from volatile to non-volatile memory such that a particular machine can recover from a crash or power outage. Although we didn't implement this part of a database in Tribune, algorithms that address this concern (such as [10] and [17]) are well known in the literature and are applied in most mature databases.

2.3 Routing in the GDP

Data must be routed to Tribune in the GDP through an overlay network. Oceanstore used Tapestry [19], although a distributed hash table approach such as Chord [18] or Bamboo [15] for routing. A project is currently underway in the GDP to efficiently route to the opaque identifiers that signify the location of a log server or a CAAPI.

3 System Architecture

ARCHITECTURE PICTURES GO HERE!!! Our first design of system emulates a simplified version of Google Spanner without any **"distributed transaction"**. Because we opt not to handle **"distributed transactions"**, we do not implement the transaction manager which coordinates two phase commits between different Paxos groups. Our first design of system assumes an one-paxos-group environment so we only implement the lock manager at the leader replica.

The responder receives transactions from client applications. It parses each operation line by line and acquires read locks for all data required for the transaction. The responder buffers writes locally. When all the transaction operation finishes, the responder tries to acquire write locks for the locally modified data. If the attempt turns out successful, the responder will try to commit the transaction at the leader.

We only implement the necessary components for read-write transactions. So all transactions would go through the leader replica to validate their respective lock leases before committing via paxos protocol. If the validation is successful, the leader would go through all paxos phases and return the final transaction status (abort or commit) back to responder. The responder will return the transaction status back to client apps.

Below is a diagram for the overall architecture:

//then talk about modifications on the first version to incorporate optimistic concurrency control and byzantine agreement

4 Algorithms and Implementation

In this section, we're going to elaborate on the algorithmic part of the project, which is our choices of concurrency control protocol and commit consensus protocol. We did strict two-phase locking and optimistic concurrency for concurrency control. We also did paxos and byzantine agreement for commit consensus protocol. We would explain in detail how we implement four algorithms in our project and compare the tradeoffs in this section. The benchmark results would be in the next section.

4.1 Strict Two-Phase Locking

```
// Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
   public void paintComponent(Graphics g) {
        g.drawString("Hello, world!", 65,
            95);
    }
}
```

4.2 Optimistic Concurrency Control

Subsubsection text here.

4.3 Pseudo Paxos vs Pseudo Byzantine Agreement

We had to deal first-hand with a non-obvious implementation detail that arises in the practical implementation of Paxos: out-of-date replicas. We took inspiration from Chubby's solution to the problem [2] by using sequence numbers to catch a replica back up to the current state of the system when possible. It would be impractical to maintain a permanent map between paxos sequence numbers and changes to the database, so instead we truncate the log at a fixed distance in the past. If a replica is further behind than the end of the log, it asks for a snapshot of the current state of the system from the leader, and achieves up-to-date status.

Also, we considered using Raft [11] as an alternative to Paxos We implement the

4.4 Remote Method Invocation

5 Experiments and Results

Subsubsection text here.

6 Future Work

Spanner-style semantics concerning time are most certainly not the only option for a CAAPI in the GDP. PTIDES [20] provides both an execution model and a simulation environment for time-aware computation. Rather than dealing with transactions, PTIDES determines the execution of atomic events and enforces that sensor-to-actuator deadlines are met. As an advantage of tight time synchronization and atomic events, PTIDES can define a deterministic time-ordered merge between two input streams by simply waiting out the clock uncertainty before writing. This policy guarantees no future event could be timestamped further in the past than the written value.

7 Acknowledgements

Thanks to Dr. Patricia Derler for her input on Spanner and time synchronization. Also thanks to Prof. John Kubiatowicz for numerous discussions about the GDP and CAAPIs.

References

[1] J. Blmer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An xor-based erasure-resilient coding scheme, 1995.

[2] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 398–407. ACM, 2007.

[3] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, and P. Hochschild. Spanner: Googles globally-distributed database. In *Proceedings of OSDI*, volume 1, 2012.

[4] P. Dabbelt and J. D. Kubiatowicz. A case for the universal dataplane, September 2013. Presented at theFirst International Workshop on the Swarm at the Edge of the Cloud (SEC'13 @ ESWeek).

[5] J. D. Kubiatowicz. Enabling the swarm through the global data plane, November 2014.

[6] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[7] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[8] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[9] E. A. Lee, J. D. Kubiatowicz, J. M. Rabaey, A. L. Sangiovanni-Vincentelli, S. A. Seshia, J. Wawrzynek, D. Blaauw, P. Dutta, K. Fu, and C. Guestrin. The TerraSwarm research center (TSRC)(a white paper). Technical report, Technical report UCB/EECS-2012-207, 2012.

[10] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems (TODS)*, 17(1):94–162, 1992.

[11] D. Ongaro and J. Ousterhout. *In search of an understandable consensus algorithm (extended version)*. 2014.

[12] J. M. Rabaey. The swarm at the edge of the cloud-a new perspective on wireless. In *VLSI Circuits (VLSIC), 2011 Symposium on*, pages 6–8, 2011.

[13] R. Ratzel and R. Greenstreet. Toward higher precision. *Communications of the ACM*, 55(10):38–47, 2012.

[14] S. C. Rhea, P. R. Eaton, D. Geels, H. Weatherspoon, B. Y. Zhao, and J. Kubiatowicz. Pond: The OceanStore prototype. In *FAST*, volume 3, pages 1–14, 2003.

[15] S. C. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. *Handling churn in a DHT*. Computer Science Division, University of California, 2003.

[16] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II. System level concurrency control for distributed database systems. *ACM Trans. Database Syst.*, 3(2):178–198, June 1978.

[17] R. Sears and E. Brewer. Segment-based recovery: write-ahead logging revisited. *Proceedings of the VLDB Endowment*, 2(1):490–501, 2009.

[18] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on*, 11(1):17–32, 2003.

[19] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, Jan. 2004.

[20] J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler. Execution strategies for PTIDES, a programming model for distributed embedded systems. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, pages 77–86. IEEE, 2009.