Improving Scalability by Self-Archiving

Zhiwu Xie[1, 2], Jinyang Liu[3], Herbert Van de Sompel[4], Johann van Reenen[1], Ramiro Jordan[1]

[1]University of New Mexico
Albuquerque, NM 87131
{zxie, jreenen, rjordan}@unm.edu

[2]George Mason University
Fairfax, VA 22030
zxie2@gmu.edu

[3]Howard Hughes Medical Institute
Ashburn, VA 20147
liuj@janelia.hhmi.org

[4]Los Alamos National Laboratory
Los Alamos, NM 87544
herbertv@lanl.gov

ABSTRACT
The newer generation of web browsers supports the client-side database, making it possible to run the full web application stacks entirely in the web clients. Still, the server side database is indispensable as the central hub for exchanging persistent data between the web clients. Assuming this characterization, we propose a novel web application framework in which the server archives its database states at predefined periods then makes them available on the web. The clients then use these archives to synchronize their local databases. Although the main purpose is to reduce the database scalability bottleneck, this approach also promotes self-archiving and can be used for time traveling. We discuss the consistency properties provided by this framework, as well as the tradeoffs imposed.

Categories and Subject Descriptors
H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Web-based services;* H.2.4 [**Database Management**]: Systems – *Concurrency, Distributed databases, Transaction processing.*

General Terms
Design, Performance.

Keywords
Database scalability, replication control, freshness, serializability, eventual consistency, client-side, archive, Memento.

1. INTRODUCTION
A Memento is a web resource that represents an archived snapshot of the original, time-varying resource. The Memento framework [1] establishes an HTTP based model and syntax to link the existing Mementos and their Original Resources, enabling time travel on the web.

The primary source for Mementos should be self-archiving. For data-driven applications, a large body of research exists on temporalizing the data model [2], and many database products have long supported archiving and time travel at the transaction level. But in practice, most web applications assume the users are only interested in the current state of the affairs. For these applications, creating and maintaining Mementos is less mission-critical than improving the performance of the current-state transactions. In this paper we present a web framework in which self-archiving is by design in the core of the current-state web application functionalities to alleviate the scalability bottleneck at the server side database. This provides a more pertinent motivation for the web applications to self-archive.

2. FRESHNESS FOR SCALABILITY
From the web client's viewpoint, the data freshness is never promised by the current-state web applications. Even if the database explicitly guarantees it, the originally fresh data must still travel across the WAN latency to reach the clients, during which period the database state could have already changed. This is even more evident when the server load is high, the application state is updated very frequently, and the database starts to choke.

Since the web clients always have to tolerate some staleness, it makes a relatively small difference for them to switch from waiting for the fresh data to become stale to immediately reusing the readily available stale data. The latter has the obvious benefit of bypassing the server side database queries, leading to much better scalability. The challenge, however, is how to trade freshness for scalability in a consistent and efficient manner.

3. SELF ARCHIVING
We start by temporalizing the data model. With a temporal database on the server, we now require the database query issued explicitly with a time predicate, therefore is conceptually against an archived database snapshot taken at a past instant. This forms a database query parallel of the Memento framework. We then limit the time predicates in these queries to a predefined set by establishing a TimeMap that all the web clients must obtain upon initialization and possibly update later. In this way, the same database snapshot can be reused by all the queries issued by different web clients that although not issued at the same instant, but falling in the same time period.

We use the motion pictures as an analogy to help understand this method. When shooting motion pictures, no matter how fast the object moves (or the database updates), the video camera takes snapshots at its own predefined frequencies. Given the archived snapshots, a video player understands how the frames are timed and assembled, and uses this knowledge to replay the snapshots to the clients (clients query against the snapshots). The states in between the snapshots are invisible to the clients, and are presumed as not worth knowing.

The above changes would have brought little relief to the scalability bottleneck if the server side database still needs to handle all the web queries. We instead take advantage of the recently popular client side database to replicate the archived database snapshot to the clients and then execute the timed queries on the clients. Since multiple clients would need the same database snapshot in approximately the same time period, its materialized view can be efficiently cached by the web intermediaries and reused, presenting a good temporal locality. Unlike queries, database updates don't need time predicates, and

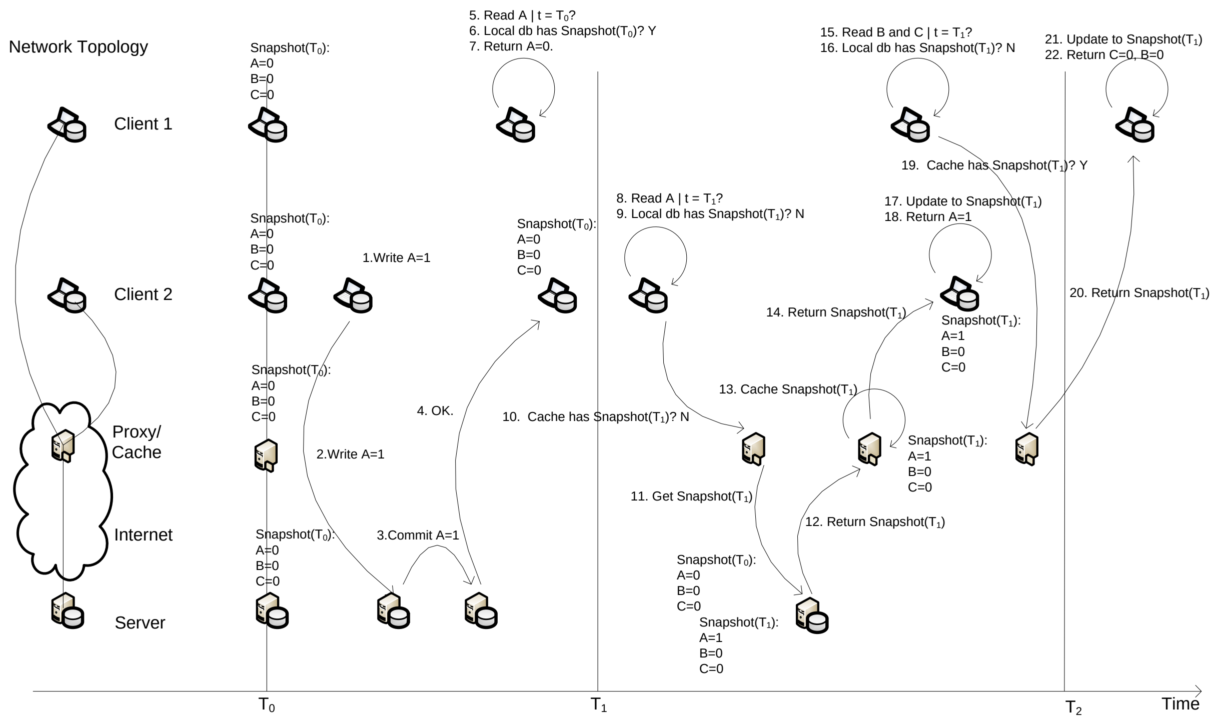

Figure 1. Improving scalability by self-archiving.

are always executed on the server side database and timestamped with their respective commit times.

Based on the above characterizations, we devise a lazy-master style [3] database replication control algorithm and can prove its correctness. The distributed execution of this algorithm is 1-copy serializable [4]. It presents a single-node view equivalence to the web clients, an important correctness property not present in the popular "eventual consistency" model [5]. We leave the details and the proof to a separate paper.

In Figure 1 we schematically show how the server side temporal database, its snapshot archives, the web proxy/cache, and the clients and their local databases are concerted by our algorithm to read and write data over the time. We note a few interesting phenomenon. First, although step 3 commits before step5, Client 1 is not aware of the updated value of A until after T_1. Second, in step 13 Snapshot(T_1) is brought in to the cache by Client 2, but is shared with subsequent queries from Client 1. Since the cache is closer to the clients than the origin server, Client 1 may get faster responses, potentially compensating the prolonged staleness.

The archiving schema can be further optimized. Archiving full database snapshots can be expensive in terms of storage and bandwidth usage. In many cases we only need to know the data changes or the writesets since the last archive. But then to rebuild a full snapshot entirely from the writesets will require many more web requests. The combination of the full snapshots and writesets provides more flexibilities and a better balance, which may differ case by case. Such archive based query also has positive implications on the security and privacy issues encountered in the other caching and replication approaches.

4. CONCLUSION

In this paper we present an archive based web applications framework. It uses the client-side databases as the replicas of the master database on the server, ensures 1-copy serializability and improves scalability. We will report the experimental verification and quantitative measurements in the follow-up work.

5. REFERENCES

[1] Van de Sompel, H., Nelson, M. and Sanderson, R. HTTP framework for time-based access to resource states -- Memento. http://tools.ietf.org/html/draft-vandesompel-memento-00

[2] Snodgrass, R.T. *Developing Time-Oriented Database Applications in SQL.* Morgan Kaufmann, 1999.

[3] Gray, J., Helland, P., O'Neil, P. and Shasha, D. 1996. The dangers of replication and a solution. *Proceedings of the 1996 ACM SIGMOD international conference on Management of data* (Montreal, Quebec, Canada, 1996), 173-182.

[4] Bernstein, P.A., Hadzilacos, V. and Goodman, N. 1987. *Concurrency Control and Recovery in Database Systems.* Addison Wesley Publishing Company.

[5] Vogels, W. 2009. Eventually consistent. *Communications of the ACM.* 52, (Jan. 2009), 40–44.