Towards a Scalable, Distributed Metadata Service for Causal Consistency under Partial Geo-replication*

Manuel Bravo[†‡], Luís Rodrigues[†], Peter Van Roy[‡]
[†]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
[‡]Université Catholique de Louvain, Belgium
angel.bravo@uclouvain.be, ler@tecnico.ulisboa.pt, peter.vanroy@uclouvain.be

ABSTRACT

Causal consistency is a consistency criteria of practical relevance in geo-replicated settings because it provides well-defined semantics in a scalable manner. In fact, it has been proved that causal consistency is the strongest consistency model that can be enforced in an always-available system. Previous approaches to provide causal consistency, which successfully tackle the problem under full geo-replication, have unveiled the inherent tradeoff between the concurrency that the system allows and the size of the metadata needed to enforce causality. When the metadata is compressed, information about concurrency may be lost, creating *false dependencies*, i.e., the encoding may suggest a causal relation that does not exist in reality. False dependencies may cause artificial delays when processing requests, and decrease the quality of service experienced by the clients.

Nevertheless, whether is possible to design a scalable solution that only uses an almost negligible amount of metadata and it is still capable of achieving high levels of concurrency under partial geo-replication, an increasingly relevant setting, remains as a challenging and interesting open research question. This position paper reports on the on-going development of Saturn, a metadata service for geo-replicated systems, that aims at mitigating the effects of false dependencies while keeping the metadata size small (even for challenging settings as partial geo-replication).

1. INTRODUCTION

Causal consistency is an interesting consistency criteria that provides well-defined semantics and avoids certain anomalies from which an eventually consistent system may suffer. Consider a scenario, in the context of social networks, in which a user comments a photo, one can say that the comment "depends on" the photo. Under causal consistency, this dependency is always satisfied (more examples and a formal definition of causal consistency can be found in [21, 23, 16, 15]). Nevertheless, in an eventually consistent system, this dependency may be violated due to arbitrary network delays. Furthermore, causal consistency has been identified as the strongest consistency criteria that can be enforced in a scalable manner [8]. Due to these interesting characteristics, many works have proposed different techniques to enforce causal consistency in a geo-replicated scenario [23, 6, 15, 16]. Unfortunately, causal consistency requires participants, including clients in some cases, to maintain some amount of metadata. It turns out that metadata management is extremely hard [12], due to the inherent trade-offs between concurrency and metadata size. For maximum concurrency, the metadata maintained by the client may be overwhelming [15]. When the metadata is compressed, information is lost and false dependencies may be introduced [10], that may cause artificial delays when processing requests, and significantly decrease the quality of service experienced by the clients.

Previous approches have successfully designed solutions under full geo-replication, where each geo-location replicates the full set of objects. Nevertheless, due to (i) the increasing number of replicas that geo-distributed applications are forced to use in order to serve current demands [1], and (ii) the emergence of a new abstraction of datacenters which vary in storage capacity and computational power [1, 2] (mostly used to further reduce access latencies), partial geo-replication becomes relevant.

To address this new challenge, we propose Saturn, a scalable, fault-tolerant, distributed metadata service, that can be used by distributed applications to enforce causal consistency when accessing replicated data. Saturn aims at reducing the metadata that needs to be maintained without limiting concurrency. Despite the conciseness of its metadata, Saturn exploits its distributed nature, and the access patterns exhibited by clients, to diminish the impact of false dependencies and avoid unnecessary delays when processing client requests, even when clients migrate among replicas. Saturn has been specifically designed to maximize its efficiency in partial geo-replicated deployments, where replicas are spread across different locations and each client is not necessarily interested in reading or writing every data item.

Saturn is a pluggable component that can be used by distributed applications that manage replicated data, including

*This work was partially funded by the Erasmus Mundus Joint Doctorate Programme under Grant Agreement 2012-0030.

Middleware Doctoral Symposium '15, December 07-11, 2015, Vancouver, BC, Canada
 ISBN 978-1-4503-3728-1/15/12 ...$15.00.
DOI: http://dx.doi.org/10.1145/2843966.2843971.

[1]An evidence is major cloud providers like Amazon [3], Microsoft [5], and Google [19], each of which already uses from ten to hundred geo-distributed datacenters; or AT&T that has thousands of datacenters at their points-of-presence (PoP) locations.

content-sharing applications, such as social networks, or distributed cooperative applications [18], among others. SATURN is deployed using a set of cooperating servers, such that the load of different clients may be distributed for scalability and also to ensure that clients have access to nearby servers to ensure minimal latency when managing the metadata. These servers are organised in a tree that is used to propagate causal dependencies among multiple geographic locations. The propagation of metadata is optimised, in function of the location of the different data items and of the data access patterns, such that false dependencies are not created unless strictly necessary to keep the metadada size practically negligible.

We are currently implementing a prototype of SATURN. Since data stores are now a fundamental building block for distributed applications, we intend to apply SATURN to Riak KV [4], a popular key-value store. We intend to compare its performance against several alternative techniques to enforce causal consistency including (i) a centralised event ordering service, based on Kronos [17]; (ii) a solution based on vector clocks [13, 29, 26]; and (iii) an explicit dependency tracking solution, based on COPS [23]. We expect SATURN to be competitive in most scenarios and achieve substantially better results in terms of scalability in partial geo-replicated settings.

2. RELATED WORK

The support for causal consistency, to simplify the application code, and to provide congruent results to the end users, can already be found in early pioneer works in distributed systems, such as Bayou [27, 25], Lazy Replication [20], and the ISIS [11] toolkit. Bayou offers session guarantees and enforces causal order of write operations to an eventually consistency data store composed by single-machine replicas in full replication settings. Lazy Replication also ensures that all operations are applied to replicas in causal order by having clients maintaining a vector clock that captures their causal past. ISIS offers a causal multicast primitive, namely *cbcast*, which provides causally ordered message delivery for group communication. All these implementations are based on vector clocks and the size of the metadata is proportional to the member of replicas and the number of objects (or process groups) managed by the system. This section briefly describes the most relevant and influential techniques that have been proposed to implement causal consistency. We classify the techniques according to their suitability to support full or partial replication.

2.1 Full Replication

Recently, and tackling scalability challenges close to ours, multiple causally consistent geo-replicated data stores have been proposed [23, 6, 15, 29, 16].

COPS [23] explicitly tracks causal dependencies per key by leveraging client support. Updates are tagged with a list of dependencies. When a datacenter propagates an update, this is not executed in the remote datacenter until all updates in the dependencies list have been executed locally. COPS assumes a fully-replicated setting where all datacenters replicate the full set of objects. This allows them to reduce the size of dependencies lists due to the transitivity rule of the happens-before relationship [21].

Orbe [15] is a causally consistent, geo-replicated, partitioned data stored. Orbe is able to reduce the size of the metadata required to track and enforce causality by relying in a vector clock with an entry per partition in the system. Thus, any two dependencies that belong to the same partition will be aggregated in a single scalar in the vector clock.

ChainReaction [6] shows how to efficiently deal with intra-datacenter replication in a causally consistent datastore. Their solution is based on a relaxed version of chain replication [28] that allows concurrent causally consistent reads of the same replicated data. ChainReaction also provides causally consistent geo-replication by using vector clocks with an entry per datacenter.

SwiftCloud [29] is a geo-replicated datastore that provides low latency reads and writes via a causally consistent client-side cache. These caches are backed by a set of datacenters that fully replicate the key-space. Causal consistency across datacenters is implemented similarly to ChainReaction.

GentleRain [16], yet another geo-replicated causally consistent datastore, is able to provide throughput similar to an eventually consistent datastore by slightly delaying the update visibility. Their protocol relies on loosely synchronized physical clocks, and it only needs a single scalar in order to track and enforce causality. GentleRain is the first system able to avoid explicit dependency check messages, which authors claim to be the major overhead of previous solutions, such as COPS.

2.2 Partial Replication

Efficiently enforcing causal consistency under partial geo-replication settings is very challenging and remains an interesting open research question [9, 29, 23].

One could adapt solutions designed for full-replication. Unfortunately this is not always easy. For instance, COPS [23] extensively uses a prune mechanism to remove obsolete dependencies from the causal past of clients. However, this prune mechanism assumes full-replication and cannot be easily modified to support partial replication. Without pruning, the solution becomes unscalable. Notice that ORBE [15] would only partially solve this problem by bounding the size of the dependencies list to the total number of servers in the system.

The PRACTI [14] work is probably the first addressing causal consistency under partial replication. Authors propose a protocol in which updates are propagated selectively to only replicas storing the updated object. Nevertheless, metadata regarding all updates still has to be seen by all participants, in order to identify gaps in the casual history. This, and the fact that each replica has to maintain a totally ordered log of updates, limits the scalability of the system. Authors propose an optimization to the protocol by introducing the concept of *imprecise invalidations* which aggregate information regarding a group of updates.

Recently, M. Shen et al. [26] have proposed algorithms based on vector clocks to achieve causal consistency in this settings. They detail two different algorithms: *Full-Track* and *Opt-Track*. The former is capable of optimally track and enforce causal dependencies without introducing false dependencies. The latter further optimizes the *Full-Track* algorithm by reducing the amortized complexity of both message size and space, achieving a promising amortized message size of $\mathcal{O}(n)$, where n is the number of sites available for clients to issue operations. Nevertheless, the message size upper bound complexity remains $\mathcal{O}(n^2)$ in both, which may substantially impact the algorithms performance.

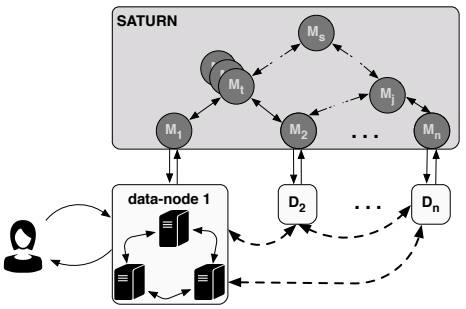

Figure 1: SATURN architectural diagram.

Finally, Charcoal [13] uses a vector clocks with an entry per datacenter in order to enforce causal consistency. It only requires to disseminate updates to datacenters that replicate the updated object by relying on loosely synchronized physical clocks, heartbeats and FIFO links. Nevertheless, their solution still poses some drawbacks: (i) the performance of the system would directly depend on the clock skew between physical clocks; and (ii) heartbeats are exchanged among all participants.

3. OVERVIEW OF SATURN

As one could infer from §2, there is a inherent tradeoff between the amount of metadata used to enforce causal consistency, and the amount of false dependencies introduced, which directly restrict the concurrency allowed by the system. For instance, systems based on COPS [23], may potentially generate large number of dependencies, which would negatively impact system's performance due to: (i) the expensive dependency checking mechanisms [16], and (ii) the message overhead ($\mathcal{O}(o)$, where o is the total number of objects). Nevertheless, COPS almost does not introduce false dependencies, which positively avoids introducing artificial delays. On the other hand, solutions such as SwiftCloud [29], Charcoal [13] and Orbe [15] reduce the message overhead ($\mathcal{O}(n)$, $\mathcal{O}(n)$ and $\mathcal{O}(n*s)$ respectively, where n is the number of replicas, and s is the number of servers that compose a replica) by compressing the metadata. Nevertheless, they introduce a large amount of false dependencies, which may reduce systems' level of concurrency. Thus, we are concern with the following question:

Is it possible to have a message overhead upper bound of $\mathcal{O}(1)$ and still be able to achieve high levels of concurrency even for challenging settings as partial geo-replication?

SATURN aims at demonstrating that such a solution is achievable. SATURN is a scalable, fault-tolerant, distributed metadata service for causal consistency specifically design to scale under partial geo-replication. It is designed to be a pluggable component that can effortlessly be integrated with distributed applications that replicate data, such as online social networks and distributed cooperative collaborative applications. Its main features can be listed as follows: (i) small and constant amount of metadata in order to track and enforce causal consistency; (ii) separated channels for metadata and data; (iii) SATURN relies on a set of cooperative servers forming a tree for the propagation of the metadata across replicas; and (iv) techniques for reducing the impact of false dependencies.

We assume a system composed of a set of clients that interact through a distributed application (Figure 1 shows how all parties interact). This application is distributed in multiple locations, namely *data-nodes*, spread across the world. Each *data-node* stores a subset of the objects. Each client has a preferred *data-node* through which it interacts with the application. Clients can issue two types of operations: updates and reads. Updates are always satisfied locally. Nevertheless, reads may query data that is not stored in client's preferred *data-node*. In this case, the read is forwarded to a remote *data-node*.

3.1 Interface

Data-nodes of a distributed application make use of SATURN in order to know when updates, originally issued in other *data-nodes*, can be executed locally without violating causal consistency. Plus, in partial geo-replicated settings, SATURN can also be used to satisfy remote read operations in a causally consistent manner.

Each update is identified by a *label* composed of the *object identifier* and a *logical time*. The *logical time* contains the identifier of the *data-node* that originally received the update and a scalar that behaves as a Lamport clock [21]. *Data-nodes* are responsible for inputting *labels* into SATURN whose task is to output them in an order that satisfies causal consistency. Notice that not all *labels* are necessarily outputted to all *data-nodes* due to the partial geo-replication setting. *Labels* are the only metadata that is propagated throughout the system.

A distributed application integrated with SATURN works as follows:

- *Write requests:* The *data-node* takes the following steps: (i) applies the update locally (if it stores the object); (ii) replies to the client, (iii) generates the update's *label*, (iv) inputs the *label* to SATURN; and finally (v) propagates the update, tagged with the *label*, to other interested *data-nodes* (this step may be performed lazily);
- *Read Requests:* If the object being queried is stored locally, the *data-node* simply returns the most recent value to the client. Otherwise, a remote read request is issued. A *data-node* only serves a remote read after ensuring that causality is not being violated. SATURN indicates *data-nodes* when serving a remote read is safe.
- *Remote updates:* A remote update is applied locally to the *data-node* when: (i) SATURN has delivered the associated *label*, and (ii) all updates associated with previously delivered *labels* have already been locally applied.

3.2 Design

We have taken a look at SATURN from the outside. Nevertheless, previous subsection only describes how distributed applications interact with the service, offering no insights into the internals. Since *labels* do not encode any information regarding order of updates across objects, SATURN internals have to still be able generate a causally ordered stream of *labels* to be delivered at each *data-node*.

We propose the use of a tree for propagating *labels* (Figure 1), where nodes are a set of cooperative servers, namely *metadata-nodes*, leaves connect to *data-nodes*, and links between nodes are FIFO. *Metadata-nodes* propagate *labels* throughout the system and deliver them to *data-nodes*. *Metadata-nodes* may fail increasing staleness of data and even partially disabling remote reads. In order to make SATURN robust, *metadata-nodes* can be replicated using stan-

dard techniques such as Paxos [22], or primary-backup [7].

A tree allows us to have a topology that we can distribute geographically, in order to benefit from the distributed nature of the applications. Thus, when choosing the topology of the tree, we can prioritize the optimization of paths between *data-nodes* that communicate frequently. Two *data-nodes* communicate due to two reasons: (i) overlapping in the objects they replicate; or (ii) remote reads.

To reduce metadata size, SATURN delivers a single stream of *labels* to each *data-node*, and the metadata is not sufficient to identify concurrency. Therefore, SATURN cannot avoid the creation of false dependencies. However, in SATURN, a false dependency only impacts negatively in the two following scenarios:

- *Scenario A*: A *label* of an update arrives before its data. This potentially delays non-causally related operations that are ordered afterwards whose *label* and data are already in the destination *data-node*. This may (i) increase the latency of remote read operations, and (ii) increase the staleness of served data.
- *Scenario B*: The data of one update arrives before its *label*. In this scenario, we are unnecessarily delaying the visibility of updates, which increases the staleness of data.

Therefore, to mitigate the impact of false-dependencies, SATURN aims at synchronizing the arrival of both *labels* and data. Thus, the system would reduce the impact of false dependencies, optimizing remote read latencies and data freshness. For this purpose, the topology of the tree in which SATURN is based, the geographical distribution of the *metadata-nodes*, and potential artificial propagation delays among *metadata-nodes* play a fundamental role.

4. FUTURE WORK

We expect this work to produce the following contributions and results: (i) the design of a scalable, fault-tolerant, distributed metadata service for causal consistency supporting partial geo-replication; (ii) the implementation of a prototype comprising our ideas, namely SATURN; and (iii) an extensive evaluation of SATURN under realistic workloads and deployments. This includes the integration of SATURN with real distributed applications, such as Riak KV [4] and its comparison to state-of-the-art solutions.

So far, we have created a simulation of SATURN for the PeerSim [24] simulator and run some initial experiments as proof of concept. In the following months, we intend to complete an implementation of SATURN and its integration with Riak KV. We will then carry out an extensive evaluation under realistic workloads and deployments. We plan to submit the work at that point to a matching conference. After that, we plan to keep researching in this direction by adding dynamism to the system and integrating self-adaptive techniques in order to create a system responsive to changes in the environment.

5. REFERENCES

[1] http://www.ericsson.com/news/121221-er-enabling-network-embedded-cloud_244159017_c.

[2] http://www.nanodatacenters.eu.

[3] http://aws.amazon.com/about-aws/global-infrastructure/.

[4] http://basho.com/products/riak-kv/.

[5] http://www.microsoft.com/en-us/server-cloud/cloud-os/global-datacenters.aspx.

[6] S. Almeida, J. a. Leitão, and L. Rodrigues. Chainreaction: A causal+ consistent datastore based on chain replication. EuroSys, 2013.

[7] P. A. Alsberg and J. D. Day. A principle for resilient sharing of distributed resources. ICSE, 1976.

[8] H. Attiya, F. Ellen, and A. Morrison. Limitations of highly-available eventually-consistent data stores. PODC, 2015.

[9] P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. The potential dangers of causal consistency and an explicit solution. SoCC, 2012.

[10] R. Baldoni and M. Klusch. Fundamentals of distributed computing: A practical tour of vector clock systems. *IEEE Distributed Systems Online*, 3(2):–, Feb. 2002.

[11] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Trans. Comput. Syst.*, 9(3), Aug. 1991.

[12] M. Bravo, N. Diegues, J. Zeng, P. Romano, and L. Rodrigues. On the use of clocks to enforce consistency in the cloud. *IEEE Data Eng. Bull*, 38(1):18–31, 2015.

[13] T. Crain and M. Shapiro. Designing a causally consistent protocol for geo-distributed partial replication. In *PaPoC*, 2015.

[14] M. Dahlin, L. Gao, A. Nayate, A. Venkataramana, P. Yalagandula, and J. Zheng. Practi replication. *NSDI*, 2006.

[15] J. Du, S. Elnikety, A. Roy, and W. Zwaenepoel. Orbe: Scalable causal consistency using dependency matrices and physical clocks. SoCC, 2013.

[16] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel. Gentlerain: Cheap and scalable causal consistency with physical clocks. SoCC, 2014.

[17] R. Escriva, A. Dubey, B. Wong, and E. G. Sirer. Kronos: The design and implementation of an event ordering service. EuroSys, 2014.

[18] V. Grishchenko. Citrea and swarm: Partially ordered op logs in the browser: Implementing a collaborative editor and an object sync library in javascript. PaPEC, 2014.

[19] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined wan. SIGCOMM, 2013.

[20] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing high availability using lazy replication. *ACM Trans. Comput. Syst.*, 1992.

[21] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7), July 1978.

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

[23] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual: Scalable causal consistency for wide-area storage with cops. SOSP, 2011.

[24] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. P2P, 2009.

[25] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. SOSP, 1997.

[26] M. Shen, A. D. Kshemkalyani, and T.-y. Hsu. Causal consistency for geo-replicated cloud storage under partial replication. IPDPSW, 2015.

[27] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. Welch. Session guarantees for weakly consistent replicated data. In *IEEE PDIS*, 1994.

[28] R. Van Renesse and F. B. Schneider. Chain replication for supporting high throughput and availability. In *OSDI*, 2004.

[29] M. Zawirski, N. M. Preguiça, S. Duarte, A. Bieniusa, V. Balegas, and M. Shapiro. Write fast, read in the past: Causal consistency for client-side applications. Middleware, 2015.