UNOBTRUSIVE DEFERRED UPDATE STABILIZATION FOR EFFICIENT GEO-REPLICATION

Chathuri Gunawardhana, Manuel Bravo, Luís Rodrigues
Problem

In a geo-replicated system, how to apply efficiently, remote updates, in causal order?
Why causal consistency?

Limitations of Highly-Available Eventually-Consistent Data Stores

Hagit Attiya  
Computer Science Department  
Technion  

Faith Ellen  
Department of Computer Science  
University of Toronto  

Adam Morrison  
Computer Science Department  
Technion  

ABSTRACT
Modern replicated data stores aim to provide high availability, by immediately responding to client requests, often by implementing objects that expose concurrency. Such objects, for example, multi-valued registers (MVRs), do not have sequential specifications. This paper explores a recent model for replicated data stores that can be used to precisely specify causal consistency for such objects, and liveness properties like eventual consistency, without revealing details of the underlying implementation. The model is used to prove the following results:

- An eventually consistent data store implementing MVRs cannot satisfy a consistency model strictly stronger than observable causal consistency (OCC). OCC is a model somewhat stronger than causal consistency, which captures executions in which client observations can use causality to infer concurrcency of operations. This result holds under certain assumptions about the data store.
- Under the same assumptions, an eventually consistent and causally consistent replicated data store must send messages of unbounded size: if $s$ objects are supported by $n$ replicas, then, for every $k > 1$, there is an execution in which an $\Omega(n \min(n,s)k)$-bit message is sent.

of data (i.e., accesses to data return without delay), and its consistency, while tolerating message delays. The CAP theorem [8,18] demonstrates the difficulty of achieving this balance, showing that strong consistency (i.e., atomicity) cannot be satisfied together with high availability and Partition tolerance.

One aspect of data consistency is a safety property restricting the possible values observed by clients accessing different replicas. The set of possible executions is called a consistency model. For example, causal consistency [2] ensures that the causes of an operation are visible at a replica no later than the operation itself. (A precise definition appears in Section 3.) A smaller set of possible executions means that there is less uncertainty about the data. So, one consistency model is strictly stronger than another if its executions are a proper subset of the executions of the other.

Some weak consistency models can be trivially satisfied by never updating the data. Therefore, another aspect of data consistency is a liveness property, ensuring that updates are applied at all replicas. The designers of many systems, e.g., Dynamo [13] and Cassandra [1], opt for a very weak liveness property called somewhat confusingly, eventual consistency [5,9,10,29]. Eventual consistency only ensures that each replica eventually observes all updates to the object, a property also referred to as update propagation [11].
Why causal consistency?

Key ingredient of several consistency criteria

- Parallel Snapshot Isolation [SOSP’11]
- RedBlue Consistency [OSDI’12]
- Explicit Consistency [EuroSys’15]
- Session guarantees [SOSP’97]
Dan is in the hospital!
Dan is ok!
That’s great!

Bob

Alice
Dan

Data center

Requires maintaining and exchanging metadata!
Metadata

Needs a way to compress metadata!

more metadata

less metadata

precise

false positives

expensive

cheap
Metadata compression

Two main ways to compress and manage metadata

Global stabilization procedures

Serializers
Metadata compression

Global stabilization procedures

Updates are propagated concurrently and later ordered at remote datacenters
Metadata compression

Serializers

Updates are ordered before being applied at the origin datacenter
**Metadata compression**

**GentleRain** and **Cure** use **global stabilization**!

GentleRain and Cure use **global stabilization**!

- **GentleRain** favors **visibility** latencies by using a vector with an entry per dc.
- **Cure** favors **throughput** by using a single scalar.

Global stabilization forces designers to either favor **throughput** or **visibility**.

- **Visibility latencies** vs. **Clock computation interval** (ms).
- **Thput (%)** vs. **Clock computation interval** (ms).

The graphs show the relationship between visibility latencies and clock computation interval, as well as throughput and clock computation interval.
Metadata compression

**Serializer** (typically one per dc): SwiftCloud [Middleware’15], ChainReaction [EuroSys’13], …
Metadata compression

**Serializer** (typically one per dc): SwiftCloud [Middleware’15], ChainReaction [EuroSys’13], …

- Serializer abstracts the complexity of partitioned datacenters, which enables trivial dependency checking thus, allowing sequencer-based systems to only slightly add artificial delays
- Unfortunately, coordination is on the client’s critical path, which significantly penalises throughput (16%)
- Sequencers can easily be overloaded for medium-size clusters. In our experiments, a max of 48 kops/s
Our goal

To achieve the latency of a sequencer and the throughput of GentleRain
Eunomia

aims at finding a novel way of compressing metadata that allows to pick a better spot in the throughput-visibility tradeoff
Eunomia

conceived to replace sequencers in geo-replicated storage systems

totally orders—consistently with causality—local updates, before shipping them to other dcs

the ordering is done in the background, out of client’s critical path
remote datacenters

A
B
C

Sequencer

16
remote datacenters

Eunomia

A

B

C

17
EunomiaKV: supporting geo-replication
EunomiaKV: supporting geo-replication

abstract partitioning and replication details from clients
EunomiaKV: supporting geo-replication
EunomiaKV: supporting geo-replication
EunomiaKV: supporting geo-replication
EunomiaKV: supporting geo-replication

receives remote updates, coming from remote Eunomia services
EunomiaKV: supporting geo-replication

only propagates to local partitions when dependencies are satisfied
Implementation

**Eunomia** is implemented in C++ (200 LOC). At its core, it uses a **red-black tree**

**EunomiaKV** is built as a **variant of** the open source version of **Riak** (100 lines of Erlang code)

Our implementation uses **hybrid logical clocks** (HLC) becoming resilient to both clock and workload skew
Evaluation: EunomiaKV vs. state-of-the-art

Three datacenter deployment, emulating Amazon EC2 Virginia, Oregon, and Ireland regions

**baseline:** system that adds no overhead due to consistency management

comparison to: global stabilisation solutions, GentleRain (single scalar) and Cure (vector as EunomiaKV)
Evaluation

Throughput comparison

EunomiaKV barely penalizes throughput when compared to the baseline.

Global stabilisation has a significant cost, specially in Cure, that uses more metadata.

Eventual

GentleRain

EunomiaKV

Cure

Throughput (ops/sec)

Global stabilisation has a significant cost, specially in Cure, that uses more metadata.
Evaluation

**Remote update visibility**

- **GentleRain** significantly penalizes visibility due to the amount of false positives.
- **EunomiaKV** provides even better results than **Cure**, using the same amount of metadata.

### CDF Graph

- **GentleRain**
- **Cure**
- **EunomiaKV**

**Remote update visibility (milliseconds)**

0 20 40 60 80 100 120

0 0.2 0.4 0.6 0.8 1
Evaluation

maximum throughput achievable by Eunomia vs a classical sequencer
Going back to the beginning
Going back to the beginning

EunomiaKV only adds a slight artificial delay, matching the latency observed in sequencer-based systems.

EunomiaKV adds negligible throughput penalty.
take-away message

by taking the coordination with an ordering service out the the client’s critical path, one can pick a sweet-spot in the throughput vs. visibility tradeoff

check the paper!!

fault-tolerant version of Eunomia
impact of stragglers
more experiments