SyncFree

SyncFree Technology White Paper

Antidote: the highly-available geo-replicated database with strongest guarantees

Deepthi Devaki Akkoorath, Annette Bieniusa
Tech. U. Kaiserslautern

23 August 2016

Why Antidote

Cloud-scale applications need highly available, low latency responses to serve millions of users around the world. To meet this need, applications have to carefully choose a high performance distributed database. Traditional databases provide strong guarantees but are slow and unavailable under failures and network partition. Hence, they are not suitable for geo-replication. The alternatives are NoSQL-style databases which are fast and available even under network partition. They provide a low-level key-value interface and expose data inconsistencies due to asynchronous communication among the servers. It takes significant effort and expertise from programmers to deal with these inconsistencies and develop correct applications on top of these databases.

For example, consider that your application stores a counter which counts the number of ads displayed to a user. For scalability, the database replicates all data in different locations. What will be the value of the counter, when it is incremented at two locations at the same time? As an application programmer, you have to detect such concurrent updates and resolve conflicting modifications.

Features of Antidote

Antidote [1] provides features that aid programmers to write correct applications, while having the same performance and horizontal scalability as AP/NoSQL, from a single machine to geo-replicated deployments, with the added guarantees of Causal Highly-Available Transactions, and provable absence of data corruption due to concurrency.

CRDT support Antidote supports high-level replicated data types [5] such as counters, sets, maps, and sequences that are designed to work correctly in the presence of concurrent updates and partial failures.

Geo-replication Antidote is designed to run on multiple servers in geo-distributed locations. To provide fast responses to read and write requests, Antidote automatically replicates data in different locations and serves the requests from the nearest location without contacting a remote server. It provides continuous functioning even when there are failures or network partition.

Partial replication supports smaller data centers near the edge, supplementing the fully-replicated data centers in the network core.

Highly Available Transactions In some cases, the application needs to maintain some relation between updates to different objects. For example, in a social networking application, a reply to some post should be visible to a user only after s/he observed the post. Antidote maintains such relations by providing causal consistency [4] across all replicas and atomic multi-object updates. Thus, programmers can program their application on top of Antidote without worrying about the inconsistencies arising due to concurrent updates in different replicas.

Internals of Antidote

To provide fast parallel access to different data, data is sharded among the servers within a cluster using consistent hashing and organized in a ring. A read/write request is served by the server hosting a copy of the data. A transaction that reads/writes multiple objects contacts only those servers that have the objects accessed by the transaction. This master-less design allows serving of requests even when some servers fail.

Antidote employs *Cure* [3], a highly scalable protocol, to replicate the updates from one cluster to other. The updates are replicated asynchronously to provide high availability under network partitions. Cure provides causal consistency [4] which is the highest consistency model compatible with high availability. Causal consistency guarantees that related events are made visible according to their order of occurrence, while the unrelated events (events that occurred concurrently) can be in different order in different replicas. For example, in a social networking application a wall post has happened before a reply to it. Therefore no user must see the reply before the post itself. Causal consistency provides these guarantees.

Cure also allows applications to pack reads and writes to multiple objects in a transaction. The transactions together with causal consistency helps to read and update more than one object in a consistent manner.

Using Antidote

The client libraries provide an easy to use API for connecting to an Antidote instance and accessing the data. For

example, a replicated counter is incremented by an Erlang client [2] as follows.

First a connection is initiated to the Antidote data center using its address and port.

```
{ok, Pid} = antidotec_pb_socket:start(?ADDRESS, ?
    PORT).
```

Antidote supports different replicated data types. These object can be created and updated by the client. Here we use counter.

```
Obj = antidotec_counter:new().
Obj2 = antidotec_counter:increment(Amount, Obj).
```

BObj identifies the object in the Antidote. An object is identified by its key and bucket.

```
BObj = {Key, antidote_crdt_counter, Bucket}.
```

Every update is wrapped in a transaction. First start a transaction and then update the object.

```
{ok, TxId} = antidotec_pb:start_transaction(Pid, {
    }),
ok = antidotec_pb:update_objects(Pid,
    antidotec_counter:to_ops(BObj, Obj2), TxId),
```

We can update more objects before committing the transaction. The stored counter can be retrieved as:

```
{ok, Counter} = antidotec_pb:read_objects(Pid,
    BObj, TxId),
```

After executing all updates/reads with in a transaction, you can commit the transaction. An update is stored only if the transaction is successfully committed.

```
{ok, _} = antidotec_pb:commit_transaction(Pid,
    TxId),
```

If you have no more updates the session can be terminated.

```
_Disconnected = antidotec_pb_socket:stop(Pid).
```

References

[1] Antidote. https://github.com/SyncFree/antidote, 2016. Accessed 12 August 2016.

[2] Antidote client library. https://github.com/syncfree/antidote_pb, 2016. Accessed 12 August 2016.

[3] D. Akkoorath, A. Z. Tomsic, M. Bravo, Z. Li, T. Crain, A. Bieniusa, N. Preguica, and M. Shapiro. Cure: Strong semantics meets high availability and low latency. In *Proceedings of 36th IEEE International Conference on Distributed Computing Systems, ICDCS*, June 2016.

[4] 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. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 401–416, 2011.

[5] M. Shapiro, N. M. Preguiça, C. Baquero, and M. Zawirski. Convergent and commutative replicated data types. *Bulletin of the EATCS*, 104:67–88, 2011.