vCorfu

A Cloud-Scale Object Store on a Shared Log

Michael Wei, Amy Tai, Christopher J. Rossbach, Ittai Abraham, Maithem Munshed, Medhavi Dhawan, Jim Stabile, Udi Wieder, Scott Fritchie, Steven Swanson, Michael J. Freedman, Dahlia Malkhi

Michael Wei
NSDI 2017
March 27th, 2017
Background

Shared Logs and Consistency
Scalability and Consistency

Systems which are scalable…

yet consistent…

can be difficult to build

1 + 1 = 2
1 + 1 = 2
1 + 1 = 2
1 + 1 = ?
Shared Log Systems

Shared log systems represent a point in the design space…

which provides scalability without compromising consistency…

however, these systems make a different set of tradeoffs
Writing vs Reading

While writing to a shared log provides strong consistency and >1/2M appends/s,

To provide the strongest level of consistency, only updates are logged,

So reads are more expensive, as clients now have to read multiple updates

.5M ops/s

1 2 3 4 5 ...

increment

1 2 3 4 5 ...

increment

1 2 3 4 5 ...

increment
Improving Read Scalability

B=??

A++

B++

1 2 3 4 5 …

Clients may read unnecessary updates to service requests

There is no locality, so clients will have to jump around on the log

Clients have to do more work to figure out the results of a transaction

Read A@1

B++

1 2 3 4 5 …

1 2 3 4 5 …
vCorfu addresses read scalability by...

- **Stream materialization**, which localizes related updates and enables reads without playback.
- **Composable SMR**, which enables large state machines without forcing clients to replicate the entire state machine.
- **Lightweight Transaction Resolution**, which eliminates the need for clients to determine whether transactions in a log were aborted.

---

Stream materialization, which localizes related updates and enables reads without playback.
---

Composable SMR, which enables large state machines without forcing clients to replicate the entire state machine.
---

Lightweight Transaction Resolution, which eliminates the need for clients to determine whether transactions in a log were aborted.
vCorfu Offers Another Point in the Design Space

Different point in the design space

Better read scalability, but at a penalty to writes

We can now service more clients without consuming the entire log
...and we will show

That we can now scale shared log systems to cloud-scale data sets

Offer comparable performance to, and often outperform state-of-the-art NoSQL systems

While retaining the strong consistency benefits of a shared log
Read(address):
Read an entry from the log

Append(entry):
Append an entry to the log and return the address it was written at

Shared Log Systems are Composed of...

**Sequencer**, which issues addresses in a log

**Log replicas**, which store data in the log

**Layout**, which maps addresses in the log to log replicas
To Append to a Shared Log, Clients…

First contact the sequencer, which issues an address

Using the layout, determine which log replicas to write to

Perform a write using the address given by the sequencer
We Take the Tango [2] Approach…

Clients don’t interact with the log directly, rather, they interact with objects.

Objects are stored in virtualized logs called streams.

Entries in the stream represent updates to the object state.

Including Support For Transactions…

The system leverages the log to provide rich support for transactions.

Transactions execute optimistically on the client’s in-memory views.

And the log serves as the ground truth in case of conflicts.

```java
if (A==1) {
    A++;  
    B++;  
}
```
vCorfu Stream Store
Architecture and Design
Materialized Streams

In vCorfu, a fundamental building block is a materialized stream.

Stream replicas implement the storage for a materialized stream.

The vCorfu sequencer keeps track of the global tail as well as stream tails (global, stream).

Gx (Global Address) /y (Stream Address)
Materializing Streams

Sequencer issues global address (2) and stream address (▲1)

Write to log replica using the global address (2)

Write to stream replica using the stream ID (▲) and stream address (1)
Materializing Streams

Client must commit data to every log and stream replica

Log replicas and stream replicas only serve committed data

Extra commit message reduces append throughput

- Materialized Stream
- Stream

K Appends/s

Total Replicas

- 2
- 4
- 8
Reading From a Materialized Stream

Stream replicas contain all updates for a given stream.

This enables reading a stream by contacting only one replica.

Not having to jump from replica to replica greatly improves read performance.
Local / Remote Views

Local views enable clients to obtain in-memory objects by following updates.

Remote views enable to delegate playback to stream replicas.

Remote views keep latency constant with a heavily modified object and many clients reading.
**Transactional Execution**

We support optimistic transaction execution based on versioned object views.

The client tracks the version of each object it accesses and generates a list of modifications it will make.

```java
if (▲ == 1) {
    ◆ ++;
}
```

Read Set: V1

Write Set: V0 ++ V1
Lightweight Transaction Resolution

Send sequencer version of read and write set, address issued if read set versions are equal

This enables only the write set to be written, since we know that the read set will not have changed

And a client encountering this entry does not need to determine the read set

---

Log Tail

1
▲: G1/0
◆: G0/0

Read ▲V1
Write ▼

G2 ◆1

++ ++

++ ++
Objects can contain large amounts of state, which pose a difficulty for SMR.

They pose a burden on the log because they contain many updates.

They pose a burden for a client, which has to play all these updates in memory.
CSMR: Composing vCorfu Objects

vCorfu objects can be composed of other vCorfu objects with a pointer

Reduces playback burden by naturally dividing objects

Leverages transactional features of vCorfu

A - L
L - Z
Instead of a single map, compose a map from multiple buckets.

Most operations only need to access a single bucket.

Certain operations, like `clear()` or `size()` are more expensive with CSMR.
vCorfu vs. Cassandra YCSB
Conclusion
vCorfu Benefits
vCorfu addresses the read burden by...

**Stream materialization**, which localizes related updates and enables reads without playback.

**Composable SMR**, which enables large state machines without forcing clients to replicate the entire state machine.

**Lightweight Transaction Resolution**, which eliminates the need for clients to determine whether transactions in a log were aborted.
Special thanks to the Corfu Team: Past and Present

Original Corfu Paper
Mahesh Balakrishnan
Dahlia Malkhi
Vijayan Prabhakaran
Ted Wobber
John D. Davis

Tango
Ming Wu
Sriram Rao
Tao Zou
Aviad Zuck

Replex
Amy Tai
Michael J. Freedman
Ittai Abraham

VMware NSX
Maithem Munshed
Zeeshan Lokhandwala
Medhavi Dhawan
Jim Stabile
Kapil Goyal
Guprit Johal
Konstain Spriov
James Chang
Jim Yang
Kevin James
Anny Manzanil
Ragnar Edholm

vCorfu
Christopher J. Rossbach
Udi Wieder
Scott Fritchie
Corfu is a consistency platform designed around the abstraction of a shared log. CorfuDB objects are in-memory, highly available data structures providing linearizable read/write operations and strictly serializable transactions. CorfuDB is based on peer-reviewed research, see References.
Questions?
We Take the Tango [1] Approach...

Where clients interact with objects, and a runtime manages interactions with the log. Each object is contained within a stream, which is the set of updates for that object.

Example: Incrementing a Counter

When the client increments the counter, the runtime asks the sequencer for the next address for the stream of the given counter.

Log Tail
-1
A: -1
B: -1

Next, A?

Counter A
0

Counter B
0

Increment
Example: Incrementing a Counter

And the runtime can now write a increment record to the log replica, writing the previous stream address given in the record, known as a backpointer.
And the runtime can now write an increment record to the log replica, writing the previous stream address given in the record, known as a backpointer.
Example: Incrementing a Counter

And the runtime can now write a increment record to the log replica, writing the previous stream address given in the record, known as a **backpointer**.
Example: Reading a Counter

To read, the runtime contacts the sequencer for the latest address issued to stream A. The client then reads all the updates, traversing the backpointers.
Example: Reading a Counter

The runtime keeps all the updates in memory until the entire stream has been read.
Example: Reading a Counter

Once the entire stream is read, the runtime applies the updates and returns the new value of the counter to the client.
Example: Holes

Holes due to failed clients can be a problem – they contain no information about backpointers, and require a linear scan if encountered.
Example: Incrementing Multiple Counters

A multi-put is implemented by generating a single entry which is part of both streams.
To increment multiple counters conditionally, a transaction is created. The runtime keeps track of the read set (address or version of read objects) and the write set. At commit time, an entry with the read set and write set is written.
Introducing a New Component: The Stream Replica

In vCorfu, we add an additional component, a stream replica, which stores data indexed not on the log address, but a combination of the stream ID and the address in the stream.
Modifying an Existing Component: Sequencer now tracks Stream Addresses

We also make a modification to the sequencer so it tracks the stream addresses used as an index for the stream replicas. This is a small counter with a small amount of state.
In Corfu/Tango, the log is striped across replica sets, as described by the layout, and each replica set is replicated via chain replication.
In vCorfu, the layout also maps each stream to a stream replica, which serve materialized views of each stream.
Example: Incrementing a Counter

To append to stream A, we now obtain a global address, backpointer and stream address from the sequencer.
Example: Incrementing a Counter

Using the global (log) address, we write to the log replica.
Example: Incrementing a Counter

Then using the stream address, we write to the stream replica. Since this is the last write we will perform, we also indicate that it is okay to commit this write.
Example: Incrementing a Counter

We then broadcast commit to any replicas we have written to. Replicas only serve committed data.
Example: Incrementing a Counter

As a result, each stream replica holds only the updates for each stream, which we refer to as a materialized stream when a stream replica is available.
In vCorfu, replica sets are no longer static. Instead, we dynamically generate replica sets based on two indexes, the log address and the stream id plus stream address.
So that in this example, three different replica sets are constructed, instead of the static chain replication protocol in Corfu/Tango.
Example: Reading a Counter

With materialized streams, reading is greatly simplified. Now instead of reading backpointers in sequence, we can read an entire stream with one request.
Now we can easily update counter B without contacting multiple replicas.
Having a single replica hold all the updates for a stream allows us to delegate playback to that replica.
Example: Reading a Counter with a Remote View

With a remote view, the client doesn’t need to have the updates or the state machine in memory.
Modifying an Existing Component: Lightweight Transaction Resolution

By adding conditional address issuance, the sequencer can perform transaction resolution.

Stream Sequencer

By adding conditional address issuance, the sequencer can perform transaction resolution.
The transaction in this example reads counter B and increments counter A if counter B is equal to two.
The client performs this transaction optimistically, and requests an address only if counter B has not changed since the client accessed it. In this case, it has not, so the address is granted.
Now, clients can read this update directly, and a client trying to determine Counter A's state does not need to read counter B at all.
Example: Incrementing Multiple Counters Conditionally

In the case that another client modifies a read object, causing the optimistic view of counter B to become invalid...
Example: Incrementing Multiple Counters Conditionally

The sequencer will reject the client’s request for an address – all by doing a simple comparison (B@0 < 3).
Example: Incrementing a Counter

And the sequencer responds with the current global address and previous stream addresses, incrementing the counters for the log and the stream.