Consistent Clustered Applications with Corfu

NSX team:
Medhavi Dhawan, Gurprit Johal, Jim Stabile,
Vjekoslav Brajkovic, James Chang, Kapil Goyal, Kevin James, Zeeshan Lokhandwala,
Anny Martinez Manzanilla, Roger Michoud, Maithem Munshed, Srinivas Neginhal, Konstantin Spirov

VMware Research team:
Michael Wei, Scott Fritchie, Chris Rossbach, Ittai Abraham, Dahlia Malkhi

The NSX R&D team and VMware Research team are using Corfu to build breakthrough, auto-configurable, auto-managed clustering management tools.

1 Introduction

As the data requirements of applications have grown in size and sophistication, the age-old DBMS model, optimizing for applications with small memory to interact with a large DB, is being disrupted.

At VMware, engineering teams who build distributed software have moved in recent years away from monolithic database systems and started using NoSQL/NewSQL scale-out software tools, e.g., ZooKeeper, Cassandra [9], MongoDB [4], Kafka [2], and so on. These tools provide industrial strength stability, and have a thriving user community support. They are successful in taking care of persistence, durability of data, and data scale out performance.

Yet, while successful at moving away from centralized databases, all of these tools manage data outside the application scope, in a format and semantics chosen by the tool. They provide very opinionated support to application builders, and little help in expressing and managing the application's own concurrency and transactional activity. They store and manipulate data outside the application, and in fact, our experience has been that these tools fail to help with the development of applications with large and complex state. More concretely, developing distributed applications that have complex state still necessitates, in most cases, a DAO (Data Abstraction Object) layer. The DAO is a transactional runtime that translates application-level data abstractions into database queries. DAOs are crafted custom-made for each application. They require to translate and transform between formats and APIs, and they result in an software architecture known as the Lambda Architecture [3]: Data is fetched into and out of the DAO, transformed and translated multiple times, into different formats, and duplicated over a plethora of tools.

The experience of an application developer over Corfu is the opposite of the above.

The application builder owns the data. Corfu helps the application side maintain an ephemeral copy inside the application program, in the format and logic that the developer chooses. The data model provided for application builders is rich and has strong semantics, making application development easier and more robust. All of this is achieved while retaining the performance, scaling and high-availability properties of share-nothing platforms.

In the rest of this short paper, we give a brief overview of the unique Corfu application-side support (the reader is deferred to previous Corfu publications for its internal design). We then describe two on-going activities using Corfu inside VMware. The first one builds a DAO-less SDN management plane. The second one is a general purpose in-memory distributed object store that has the potential for significant performance and reliability gains for application developers.

2 Corfu: A Database-less Platform

> If we could start with a clean slate and design a tool that aids the developer of distributed software, what would it look like?

Most of the wisdom in Corfu is concentrated in a client-side runtime library. The Corfu runtime builds in-memory transactional objects backed by a shared log. Objects may be arbitrary Java collections, expressing the application logic precisely in the manner that is most natural to, namely, as a programming language data structure.

Corfu manages data on the application side in the form of transactional, persistent, replicated objects over a distributed shared log. Since Corfu runtime objects are

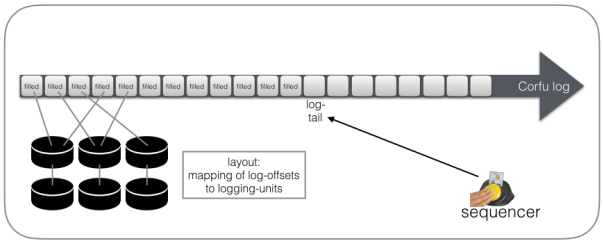

Figure 1: The Corfu distributed shared log.

backed by a log, the state of each object is represented by a history of updates.

The distributed shared log is designed to spread data over a cluster of storage nodes, each logging data in log-structure (append-only) manner. It is replicated and auto-configurable, hence all the heavy-lifting for reliability, durability, elasticity, and total-ordering are made in the log. Figure 1 gives a high level schematic of the log basic architecture: Log *append* consults a contention-remover *sequencer* about the current offset of the tail, and consults a *layout map* to translate the tail-offset to logging-units. Accompanying papers provide more detail concerning the design of Corfu and various optimizations built into it [5, 12, 13].

In memory, the Corfu runtime holds a single copy of each object. We will refer to it as a Concurrent-Versioned-Object, or in short a CVO.

Figure 2 depicts the life-cycle of a Corfu object in the runtime and on the log.

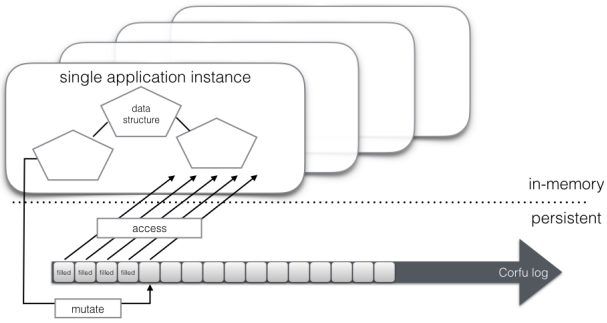

Figure 2: Transactional, persistent, replicated objects over a distributed shared log.

There are two key design properties of CVOs:

1. Lazy update: When an update record is appended to the log (or to an optimistic transactional stack, see later), it is not applied to CVOsimmediately. Only when application code accesses an object does Corfu applies updates to the CVO.
2. Single copy: Corfu optimizes for in-memory execution, hence it keeps only one version of each object at any moment in time. This is enabled by rolling CVOsforward and backward, as threads are switched, to get to their desired version.

It is worth noting that the single-copy property sets Corfu apart from traditional MVCC (multi-version concurrency control) systems. MVCC supports optimism by maintaining multiple versions until they are no longer needed (i.e., no transactions are open that needs to view these versions). There is a rich body of literature concerned with efficient arrangement of MVCC information in DBMS [6, 11] and of in-memory OLTP systems [10, 7]. However, the cost of retaining multiple versions of data would be prohibitive for the application workloads Corfu aims for, as we see below,

Not only that, in Corfu the application annotates objects to indicate when mutation to the object conflict or not. Corfu allows concurrent threads to optimistically mutate different versions of the object, which results in great flexibility and concurrency. By comparison, Microsoft's Hekaton [7] lets only one thread operate optimistically over any particular object, hence only one version is speculative at any moment in time.

If Corfu was single-threaded, things would be easy: Each time an application made an access to a Corfu object, the runtime would "play" the history of updates to synchronize the object state.

The Corfu runtime, however, provide multi-threading support, as well as supports optimistic transactions by each one of these threads.

Each time a Corfu thread performs an access operation on an object, the Corfu runtime synchronizes the object state to the thread's desired version. At any moment in time, CVO state reflects two versions: The first one is the log offset, up to which all updates were applied. The second is an optional optimistic version, an optimistic stream applied to the CVO by a speculative transaction.

Inside a transaction, bringing the CVO to the thread's desired state means synchronizing it to the transaction snapshot, and applying optimistic mutations, it any. In a non-transactional context, the desired version for an access is the current log tail, guaranteeing strict linearizability.

To get to its desired object version, a thread may need to roll back the optimistic stream of another thread. Then it needs to roll backward or forward the normal log to the desired log position. Finally, it needs to apply the optimistic updates made by its own transaction, if any.

To commit a transaction, Corfu simply needs to send a request to the sequencer. The sequencer checks for a conflict based on the transaction snapshot and the current log-tail. This is a trivial check, requiring the sequencer

to simply compare the last mutation offset to objects in the transaction read/write-set and the transaction snapshot offset. In this way, in a single round-trip, the transaction obtains a transaction resolution response, and if it can commit, a reserved offset in the log to persist the transaction record.

3 SDN Management Plane (MP)

SDNs became possible with the advent of newer generations of commodity hardware and next generation software. Instead of having networking functions like switching and routing in custom hardware, it is now possible for these networking functions to be re-thought and implemented purely in software, typically but not necessarily in a hypervisor. In a scale-out system of many hypervisors, the notion of a logical network topology (the set of logical switches, routers, firewalls, etc) is now distributed, running on many hypervisors, each doing local packet processing and forwarding. SDNs contain a separate scale-out control/management plane (controllers) that both implement networking APIs and also coordinate the programming of the forwarding engines in each hypervisor.

Rewriting the NSX control plane software (a preliminary description of which appeared in [8] over Corfu provided significant advantages which we proceed to describe.

High Availability

The NSX SDN controller exposes a RESTful API interface for managing the logical network topology. Using this API, a client can create, modify, or delete various components of the topology, including logical switches, logical routers, firewalls etc. We have migrated the controller on top of Corfu as a datastore for this logical topology. The benefit is that now the datastore is highly available with redundant copies stored in different Corfu servers (Corfu cluster). No single failure is tolerated–the SDN controller must be able to serve APIs even during partial outages or network partitions.

While traditional relational databases could be used, they are notoriously difficult and expensive for an administrator to maintain, especially as the SDN controller is shipped and installed directly by a customer. Moreover, most relational databases do not provide quorum-based replication and typically have limited scale. As long as there is a quorum of Corfu servers available, the SDN system can continue to serve APIs.

Transactional

Updates to the logical networking topology via the API has requirements for transactional behavior. In some APIs, several logical entities are manipulated in and all-or-nothing (atomic) model. For example, there are use cases when creating a new routed logical network (logical router) that require that both the logical router and a set of logical switches be created atomically.

This is where Corfu data model greatly contributed to simplicity and robustness. Most NoSQL systems provide for a document-oriented model with no transactions between documents. Corfu allows the SDN controller to be developed with individual objects (rather than documents) per logical networking element, and supports transactions to update multiple objects atomically. This is all done while retaining the desired properties of a highly-available, replicated, scale-out storage system.

Since Corfu has snapshot read semantics, that means when reading multiple objects, the SDN controller operates on a consistent set of data. Most other data platforms we considered compromise the isolation level in order to achieve scale, and default to lesser isolation levels such as read committed, which can lead to reading a mixed-version of multiple objects, even though they were updated in an atomic transaction. Using snapshot read semantics in traditional databases is typically not used because of scale and performance issues.

Programming Model

Corfu exposes plain old Java-based data structures. This allows the SDN Controller to use direct Java data structures (e.g., HashMaps) for its logic. Transactions are demarcated around access to one or more Java objects. The entire NSX control plane code-base is now expressed in a pure Java-only model, as opposed to traditional databases that require a different language (SQL) and a different model (relational tables). Our experience so far indicates that developing control plane applications using Corfu achieves simple and familiar data structures, without the object-relational impedance mismatch that traditional Java-based database applications suffer.

Scale Out

From day one, it was clear that scaling was a paramount requisite of the NSX control plane, hence data tools like ZooKeeper or etcd cannot satisfy this requirement. As the size of the logical network topology grows, so does the computation requirements when determining what information is needed to be sent to each hypervisor in order to forward network packets. Typically, as VMs (Virtual Machines) change state (e.g, power on, move

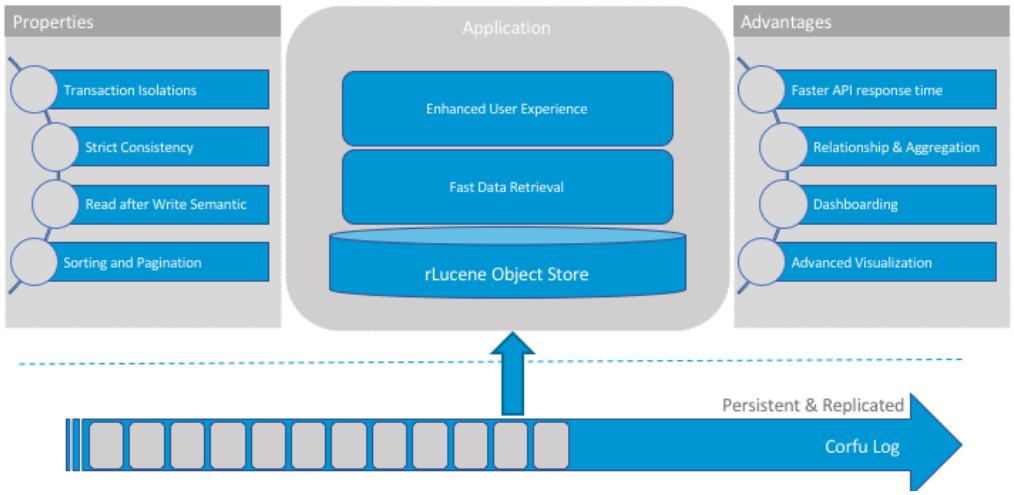

Figure 3: rLucene: An in-memory index over Corfu

to different hypervisors), the SDN controller is consulted with an updated set of forwarding configuration. In some cases (e.g., security), these VM state changes will block network VM traffic until the controller is consulted for up-to-date forwarding information. As the scale and computation complexity goes up, the latency can also increase. To combat this, the set of controllers is scaled out so that no single controller becomes a bottleneck and latency is minimized.

Corfu is designed to meet these scaling needs through virtualization of the object space into separable materialized streams [13]; by striping the history of updates across a cluster [5]; and through the single-copy concurrent runtime described here.

4 rLucene

rLucene is an in-memory object store with deep indexing for fast data retrieval. It is a fault-tolerant, highly available and fully replicated client store built on top of Corfu. Deep indexing of object data fields helps significantly in fast and flexible querying of stored objects.

rLucene is powered by the Lucene [1] search engine at the core. rLucene provides Java Map interface to abstract low level Lucene indexing and search APIs. Working with rLucene is as simple as Java Hashmap. Operations like `values()`, `keySet()` and `entrySet()` of the Java map interface are overloaded to accept queries for retrieving subsets of objects stored. rLucene map querying is blazing fast, as result-set objects are lazy-loaded. On iteration of result-set, real cost of retrieving fully populated objects is incurred.

Using Corfu for a clustering and persistence solution, rLucene provides applications the ability to store and retrieve objects from an in-memory indexed map, finely tuned for fast data retrieval. The objects are persisted in the Corfu log and are ephemeral on the client-side. It is a perfect use-case of Corfu; it opens a world of possibilities for application builders, bringing the full-fledged capabilities of Lucene indexing and search straight into the Corfu application. In rLucene, as the name suggests, the Corfu log entries are applied in sequence order to all application instances. Hence, the index is replicated because every engine instance applies the log from the beginning in the same sequence order. The strong consistency guarantees from Corfu provide read-after-write semantics. Transforming the Lucene library into a Corfu object is a great demonstration of the utility Corfu brings in convenience and expression for applications with sophisticated logic.

To understand the implementation, we will describe a `put()`. When an application `put()`s an entry, it does **not** get inserted into the in-memory map, but rather to the Corfu log. In this way, the entry is persisted to, and replicated by, Corfu. Recall that Corfu in-memory objects are "lazy", they are sync'ed with the log only when needed. In rLucene, the object is put into the in-memory Lucene engine and indexed only when a `get()` access is invoked on the map. Other map API methods like putAll(), putIfAbsent(), and so on, operate in a similar manner.

Access operations on an rLucene map are made sure to happen only after generation and visibility of new index changes. This is done to ensure read after write semantics. For this, rLucene leverages Lucenes near-real time search functionality. To make index changes visible, flipping of searcher is done in read path rather than on after every object mutation. This significantly reduces ingestion cost. To further reduce penalty on first read

operation after a change happens, a background thread regularly makes new index changes visible.

Additionally, an rLucene map has built-in support for sorting and pagination. It supports cursor-based and offset-based pagination. It is thread-safe and lock contention is kept at minimal to achieve high throughput in multi access scenarios.

Last, but not least, application builders can wrap blocks of operations as transactions. This provides multi-threaded applications with lock-free concurrent transactions. These lock-free transactions allow applications to express complex cross-index logic, such as "*if key K exists in one map, insert K′ to another map,*" while guaranteeing atomicity and isolation from other transactions. This is where the single-copy feature of Corfu has been crucial: Each transaction has its own Concurrent-Versioned-Object (CVO) snapshot of the Lucene map, with strict consistent snapshot guarantees, and without incurring the cost of cloning the entire map.

Figure 3 summarizes the rLucene architecture and features. As of this writing, rLucene is production ready, and its integration with the NSX management plane has begun.

Acknowledgement

This work would not be possible without the faith of the entire management chain and the trust they put in us to build a new data technology that drives the SDN control plane. In particular, on the NSX side, we are grateful to Rajneesh Bajpai, Ragnar Edholm, Jeff Jennings, Umesh Mahajan , Robert Tremblay; and to David Tennenhouse on the VMware Research side.

References

[1] Apache lucene. `https://en.wikipedia.org/wiki/Apache_Lucene`.

[2] Kafka. `http://kafka.apache.org/`.

[3] Lambda architecture. `https://en.wikipedia.org/wiki/Lambda_architecture`.

[4] 10GEN. MongoDB. `http://www.10gen.com/white-papers`, 2011.

[5] BALAKRISHNAN, M., MALKHI, D., DAVIS, J. D., PRABHAKARAN, V., WEI, M., AND WOBBER, T. Corfu: A distributed shared log. *ACM Trans. Comput. Syst. 31*, 4 (Dec. 2013), 10:1–10:24.

[6] BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN, N. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

[7] DIACONU, C., FREEDMAN, C., ISMERT, E., LARSON, P.-A., MITTAL, P., STONECIPHER, R., VERMA, N., AND ZWILLING, M. Hekaton: Sql server's memory-optimized oltp engine. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2013), SIGMOD '13, ACM, pp. 1243–1254.

[8] KOPONEN, T., CASADO, M., GUDE, N., STRIBLING, J., POUTIEVSKI, L., ZHU, M., RAMANATHAN, R., IWATA, Y., INOUE, H., HAMA, T., AND SHENKER, S. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 351–364.

[9] LAKSHMAN, A., AND MALIK, P. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev. 44* (April 2010), 35–40.

[10] LARSON, P.-A., BLANAS, S., DIACONU, C., FREEDMAN, C., PATEL, J. M., AND ZWILLING, M. High-performance concurrency control mechanisms for main-memory databases. *Proc. VLDB Endow. 5*, 4 (Dec. 2011), 298–309.

[11] LOMET, D. B. Key range locking strategies for improved concurrency. In *Proceedings of the 19th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1993), VLDB '93, Morgan Kaufmann Publishers Inc., pp. 655–664.

[12] TAI, A., WEI, M., FREEDMAN, M. J., ABRAHAM, I., AND MALKHI, D. Replex: A scalable, highly available multi-index data store. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference* (Berkeley, CA, USA, 2016), USENIX ATC '16, USENIX Association, pp. 337–350.

[13] WEI, M., TAI, A., ROSSBACH, C., ABRAHAM, I., MUNSHED, M., DHAWAN, M., STABILE, J., WIEDER, U., FRITCHIE, S. L., SWANSON, S., FREEDMAN, M., AND MALKHI, D. vCorfu: A cloud-scale object store on a shared log. In *NSDI* (2017).