Calvin: Fast Distributed Transactions for Partitioned Database Systems

SIGMOD ’12, May 20-24, 2012

Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, Daniel J. Abadi From Yale University

Work was sponsored by the National Science Foundation (NSF) of USA and National Natural Science Foundation of China
Agenda

• What is Calvin and some Background
  • Common issues and how Calvin helps addressing those

• Calvin – System Architecture
  • Checkpointing, Performance and Scalability

• Related Work and Future Work
  • Conclusion and Q&A
What is Calvin?

- A practical transaction scheduling and data replication layer
- It guarantees significantly reduction in normally prohibitive contention costs associated with distributed transactions
- It supports disk-based storage, scales near-linearly on a cluster of commodity machines, and has no single point of failure
- Calvin is also able to support multiple consistency levels—including Paxos-based strong consistency across geographically distant replicas—at no cost to transactional throughput
Background – Distributed Transactions

- Current trends ➔ Move away from traditional ACID database transactions
- Some systems, such as Amazon’s Dynamo, MongoDB, CouchDB, and Cassandra provide no transactional support
- Others provide only limited transactions support, such as single-row transactional updates (e.g. Bigtable) or transactions whose accesses are limited to small subsets of a database (e.g. Azure, Megastore, and the Oracle NoSQL Database)
- Other systems (e.g. VoltDB) support full ACID, but cease (or limit) concurrent transaction execution when processing a transaction that accesses data spanning multiple partitions
- Reducing transactional support ➔ simplifies the task of building linearly scalable distributed storage solutions that are designed to serve partitionable applications
- For applications that are not easily partitionable, however, the burden of ensuring atomicity and isolation is generally left to the application programmer
Background – Contd.

- Calvin is designed to run alongside a non-transactional storage system transforming it into a shared-nothing (near-)linearly scalable database system that provides high availability and full ACID transactions.
- These transactions can potentially span multiple partitions spread across the shared-nothing cluster.
- Calvin accomplishes this by providing a layer above the storage system that handles the scheduling of distributed transactions, as well as replication and network communication in the system.
The cost of Distributed Transactions

- The primary mechanism by which System (R*-style) distributed transactions impede throughput and extend latency is the requirement of an agreement protocol between all participating machines at commit time to ensure atomicity and durability.
- To ensure isolation, all of a transaction’s locks must be held for the full duration of this agreement protocol, which is typically two-phase commit.
- It requires multiple network round-trips between all participating machines, and therefore the time required to run the protocol can often be considerably greater than the time required to execute all local transaction logic.
- Allowing distributed transactions may also introduce the possibility of distributed deadlock in systems implementing pessimistic concurrency control schemes.
Consistent Replication

- A second trend has been towards reduced consistency guarantees with respect to replication (e.g. Dynamo, SimpleDB, Cassandra, Voldemort, Riak)
- The typical reason given is the CAP theorem — in order for the system to achieve 24/7 global availability and remain available even in the event of a network partition, the system must provide lower consistency guarantees
- In 2011, this trend is starting to reverse with several new systems supporting strongly consistent replication. Google’s Megastore and IBM’s Spinnaker are synchronously replicated via Paxos
- Synchronous updates come with a latency cost fundamental to the agreement protocol, which is dependent on network latency between replicas. This cost can be significant, since replicas are often geographically separated to reduce correlated failures
How Calvin addresses these issues?

- Calvin’s approach to achieve inexpensive distributed transactions and synchronous replication: when multiple machines need to agree on how to handle a particular transaction, they do it outside of transactional boundaries—that is, *before* they acquire locks and begin executing the transaction.

- Our experiments show that Calvin significantly outperforms traditional distributed database designs under high contention workloads. We find that it is possible to run half a million TPC-C transactions per second on a cluster of commodity machines in the Amazon cloud, which is immediately competitive with the world record results currently published on the TPC-C website that were obtained on much higher-end hardware.
System Architecture

Sequence Layer or Sequencer

- Intercepts transactional inputs and places them into a global transactional input sequence — this sequence will be the order of transactions to which all replicas will ensure serial equivalence during their execution. It also handles the replication & logging of this input sequence.

Scheduling Layer Scheduler

- Orchestrates transaction execution using a deterministic locking scheme to guarantee equivalence to the serial order specified by the sequencing layer while allowing transactions to be executed concurrently by a pool of transaction execution threads.

Storage Layer

- Handles all physical data layout. Calvin transactions access data using a simple CRUD interface; any storage engine supporting a similar interface can be plugged into Calvin fairly easily.

Note: All three layers can scale horizontally.
System Architecture – Contd.

Sequencer and Replication

- Calvin currently supports two modes for replicating transactional input:
  1. Asynchronous Replication
  2. Paxos-based synchronous replication
- In both modes, nodes are organized into replication groups (For example, in Fig -1 Partition 1 in Replica A and Partition 1 in Replica B would together form one replication group)
- Calvin’s current implementation uses Zoo-Keeper approach (pretty like Chubby which was proposed by Google)
- **Notes**: Zookeeper is built on a new totally ordered broadcast protocol, which is different from standard Paxos algorithm. Yahoo! Research describes the new paxos-like protocol called Zab, and show that the Zab is easy to understand, implement, and also gives high performance.
Scheduler and Concurrency Control

**Issue:** When the transactional component of a database system is unbundled from the storage component, it can no longer make any assumptions about the physical implementation of the data layer, and cannot refer to physical data structures like pages and indexes, nor can it be aware of side-effects of a transaction on the physical layout of the data in the database. Both the logging and concurrency protocols have to be completely logical, referring only to record keys rather than physical data structures.

**Solution:** Calvin could use an approach of by creating virtual resources that can be logically locked in the transactional layer (implementation remains future work)
System Architecture – Contd.

- Clients specify transaction logic as C++ functions that may access any data using the basic CRUD interface.
- Transaction code does not need to be at all aware of partitioning.
- Once a transaction has acquired all of its locks under this protocol it is handed off to a worker thread to be executed. Each transaction execution by a worker thread proceeds in five phases:
  1. Read/write set analysis
  2. Perform local reads
  3. Serve remote reads
  4. Collect remote read results
  5. Transaction logic execution and applying writes
Dependent Transactions

- Transactions which must perform reads in order to determine their full read/write sets (which we term dependent transactions) are not natively supported in Calvin. Calvin’s deterministic locking protocol requires advance knowledge of all transactions’ read/write sets before transaction execution can begin.

- Calvin supports a scheme called Optimistic Lock Location Prediction (OLLP), which can be implemented at very low overhead cost by modifying the client transaction code itself.

- The idea is for dependent transactions to be preceded by an inexpensive, low-isolation, un-replicated, read-only reconnaissance query that performs all the necessary reads to discover the transaction’s full read/write set.
Calvin’s design principle is to move as much as possible of the heavy lifting to earlier in the transaction processing pipeline, before locks are acquired.

Any time a sequencer component receives a request for a transaction that may incur a disk stall, it introduces an artificial delay before forwarding the transaction request to the scheduling layer and meanwhile sends requests to all relevant storage components to “warm up” the disk-resident records that the transaction will access.

If the artificial delay is greater than or equal to the time it takes to bring all the disk-resident records into memory, then when the transaction is actually executed, it will access only memory resident data.
Calvin With Disk-Based Storage

**Disk I/O latency prediction**
- Accurately predicting the time required to fetch a record from disk to memory is not an easy problem. The time it takes can vary significantly for many reasons:
  - Variable physical distance for the head and spindle to move
  - Prior queued disk I/O operations
  - Network latency for remote reads
  - Failover from media failures
  - Multiple I/O operations required due to traversing a disk-based data structure

**Globally tracking hot records**
- A comprehensive exploration of this strategy, including investigation of how to implement it for multi-partition transactions, remains future work.
Checkpointing

Deterministic database systems have two properties that simplify the task of ensuring fault tolerance:

◦ First, active replication allows clients to instantaneously failover to another replica in the event of a crash.
◦ Second, only the transactional input is logged—there is no need to pay the overhead of physical REDO logging.

Calvin supports three checkpointing modes:

◦ Synchronous checkpointing
◦ An asynchronous variation of Cao et al.’s Zig-Zag algorithm
◦ An asynchronous snapshot mode that is supported only when the storage layer supports full multi-versioning
Checkpointing

Although there is some reduction in total throughput due to:

◦ the CPU cost of acquiring the checkpoint; and
◦ a small amount of latch contention when accessing records, writing stable values to storage asynchronously does not increase lock contention or transaction latency.

Measurement was taken on a single machine Calvin deployment.

Figure 3: Throughput over time during a typical checkpointing period using Calvin’s modified Zig-Zag scheme.
Performance And Scalability

TPC-C New Order transactions executed per second as a function of the number of Calvin nodes, each of which stores a database partition containing 10 warehouses.

Figure 4: Total and per-node TPC-C (100% New Order) throughput, varying deployment size.
Handling High Contention

Contention index is the fraction of the total set of hot records locked by each transaction, so a contention index of 0.01 means that up to 100 transactions can execute concurrently, while a contention index of 1 forces transactions to run completely serially.

Figure shows the factor by which 4-node and 8-node Calvin systems are slowed down (compared to running a perfectly partitionable, low-contention version of the same workload) while running 100% multipartition transactions, depending on contention index.
Related Work

- Google’s Megastore and IBM’s Spinnaker recently pioneered the use of the Paxos algorithm for strongly consistent data replication in modern, high-volume transactional databases.
- Like Calvin, Spinnaker uses ZooKeeper for its Paxos implementation.
- Since they are not deterministic systems, both Megastore and Spinnaker must use Paxos to replicate transactional effects, whereas Calvin only has to use Paxos to replicate transactional inputs.
Future Work

- Even though Calvin handles hardware failure yet the replication may get slower in case of remote reads.
- In future a more seamless failover system is targeted making it invisible by leveraging replication subgroups.
- Improvement: Outgoing messages related to node A in one replica are sent not only to the intended node B but also to every replica of node B within the replication subgroup.
Conclusions

- Calvin is a transaction processing and replication layer designed to transform a generic, non-transactional, un-replicated data store into a fully ACID, consistently replicated distributed database system.

- Calvin supports horizontal scalability of the database and unconstrained ACID-compliant distributed transactions while supporting both asynchronous and Paxos-based synchronous replication, both within a single data center and across geographically separated data centers.

- By using a deterministic framework, Calvin is able to eliminate distributed commit protocols, the largest scalability impediment of modern distributed systems.
Questions?


Supplement Study/References:
5. Chubby (Google Research) http://research.google.com/archive/chubby-osdi06.pdf