Fast Logging and Recovery on Single Node with Multi-core

Youhui Bai
USTC
youhuibai0108@gmail.com

Cheng Li
USTC
chenglinkcs@gmail.com

Yinlong Xu
USTC
ylxu@ustc.edu.cn

1 Background and Motivations

Failures(e.g., process crash, kernel panics, power outages) are common in systems like Database systems and File systems, even in robust enough Distributed systems, which will incur fatal problems such as inconsistency and data loss. Techniques have been proposed to address these issues, for example, systems offten leverage Log and Replication to handle failures. In this paper, we focus on Log especially write-ahead log, systems will log changes into Log for persistency, then to the real place, When restarting, systems recovery from Log take advantage of full operation informations in Log.

But the logging when forward processing and recovery from Log after failures are both too slow to support high-level application well, which is intolerable even for a single node. the architeture and process flow of a single node is illustrated as Fig. 1. The `Disk` and `Dram` are managed by page. When transactions want to operate, they map the operations into Data buffer in step 1, then log the operations in Log buffer in step 2, the log entries in Log buffer will be flushed to durable storage for persistency in step 3, then, if the flush returns successfully in step 4, the system will inform user that the operations are completed, and the system will flush the dirty page in Data buffer into durable storage later in step 5.

Generally, systems leverage single Log to log and recover for simple, this is why the logging and revovery are so slow. First, the multiple threads will contend to write to a single Log, second, a sequential scan will be performed over the corresponding log by only one thread. Both of them will cause worse scalability as the number of threads grows in a single node with multi-core.

A straightforward approach to speed up the logging and recovery is distributing the Log, for example, one Log file per processor. But the distributed Log is still tardy because of too many cross-log dependencies.

In this paper, we propose a new version of distributed Log by leveraging a few techniques, which will support parallelism well in single node with multi-core.

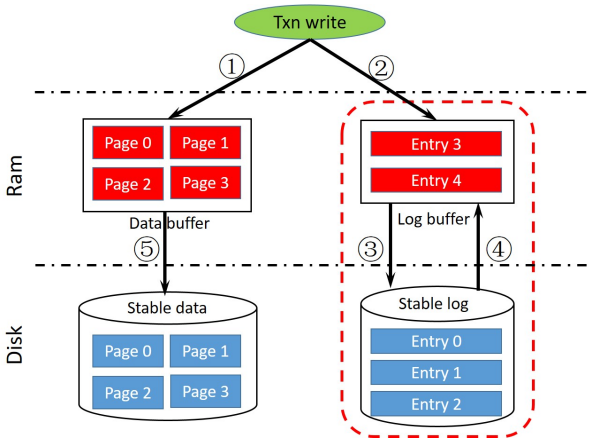

Figure 1: Architecture and process flow of write-ahead logging.

2 Our Proposal

In this section, we will propose two new techniques to eliminate the cross-log dependencies, which will accelerate the logging and recovery.

2.1 Vector Clock vs Lamport's Clock

The order is defined in [] by Lamport's clock, which is a global total order and will incur unnecessary false dependeicies. For example, the right side of Fig. 2 shows the Lamport's Clock in distributed Log, each block indicates one log entry. When H wants to commit, all of the changes before it must be flushed to disk even though some of them are indenpendent with each other, this kind of dependency is called false dependency and will incur performance slow down.

We can adopt Vector Clock to avoid the false dependency, whose order is partial, the left side of Fig. 2 illustrates the Vector Clock in distributed Log. When H

wants to commit, only the really dependent log entries are flush to disk, so we can flush part of the Logs rather than almost all of them.

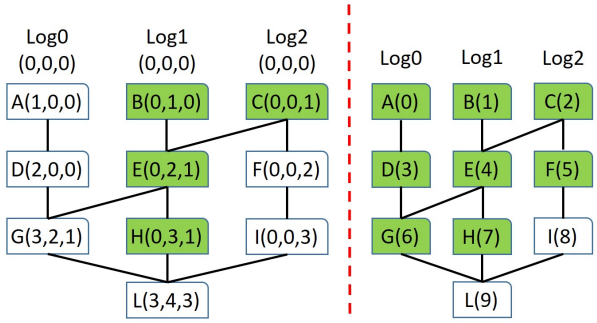

Figure 2: Vector Clock vs Lamport's Clock.

2.2 Conflict-free Replicated Data Types

We can make some conflict operations commutative by using Conflict-free Replicated Data Types(CRDTs), which means that the result will be consistent even though commute the order of related operations. We give a simple example to propose our method. In the left side of Fig. 3, transaction 2(txn2) and 4(txn4) want to modify the same object A, and txn2 operates `A = A * 2`, then txn4 operates `A = A + 4`, the final result of A is 10. But if we commute the operations of txn2 and txn4, the result will be 14 and inconsistent.

If we transfer the multiplication into addition such as the example in Fig. 3 right side, the operation `A = A * 2` can be transferred as `A = A + 3`, we can issue operations of txn2 and txn4 without order to improve performance and the result will be always consistent.

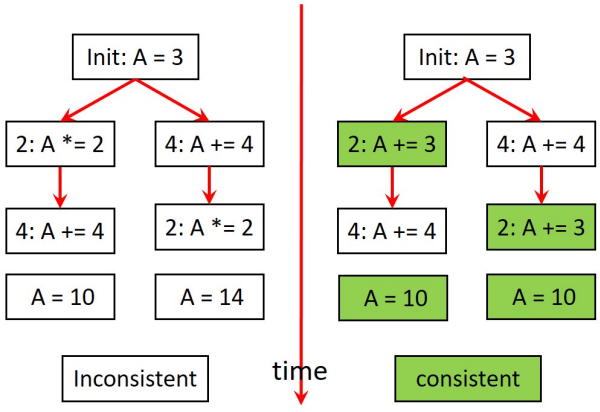

Figure 3: Correlations update.

3 Conclusion and Future Work

In this paper, we propose a new version of distributed Log by leveraging Vector Clock and CRDTs to eliminate cross-log dependencise, which can be processed by multiple threads simultaneously so as to speed up the logging and recovery and support high level applications better. We'd like to run some experiments to make our motivations steady by first, then consider the further utilization of multi-core such as CPU-affinity to optimized our distributed Log and implement our key ideas in a real system.

References