**No Compromises: Distributed Transactions with Consistency, Availability, and Performance**

The main focus of the paper is on a main memory distributed computing platform called **FaRM**. FaRM provide distributed transactions with strict serializability, incredibly high performance, low latency, durability, and high availability. In this paper a system was proposed that have protocols designed from the two hardware trends appearing in data centers fast commodity networks with RDMA and an inexpensive approach to providing non-volatile DRAM. FaRM uses one-sided RDMA operations. RDMA (Remote Direct Memory Access) supports a fast communication method that provides one-way direct access to the memory of another machine by bypassing the CPU and it avoids most local CPU overhead.

For transactions this system use one-sided RDMA reads during transaction execution and validation. FaRM do not rely on traditional mechanisms used for transaction instead it uses reservations to ensure that there is space in the logs for all the records needed to commit and truncate a transaction before starting the commit. FaRM leverages parallelism effectively thus resulting into fast recovery protocol.

FaRM uses non-volatile DRAM and fast commodity networks with RDMA to eliminate storage and network bottlenecks. The proposed system have introduced a distributed UPS integrates Lithium-ion batteries into the power supply units Instead of a centralized, expensive UPS (uninterruptible power supply). Coordinators use one-sided RDMA when logging records to non-volatile write-ahead logs at the replicas of objects modified in a transaction and transactions use no foreground CPU at backups and this CPU is used later in the background when lazily truncating logs to update objects in-place. FaRM provides strict serializability and uses optimistic concurrency. During the execution phase of a transaction, objects reads happen from memory (local access or via RDMA), and all writes are buffered locally. The address and version of every object accessed is recorded. At the end of the transaction, the commit protocol proceeds.

The commit protocol has a Lock phase in which the coordinator writes a LOCK record to the log on each machine that is a primary for any written object. In validation phase the coordinator checks whether the versions of all objects read by the transaction have been changed or not. In commit backups phase the coordinator waits for an acknowledgment from NIC after writing a commit backup record to log at each backup. In commit primaries phase the coordinator writes a commit-primary record to the logs at each primary and on receiving at least one hardware acknowledgment, completion is reported to the application. For all records primaries update objects in place, incrementing their versions, and unlocking them. In truncation phase the coordinator on receiving acknowledgements from all the primaries, truncate logs at primaries.

FaRM depends upon the information of primary and backups of every data region. All machines must agree on the membership of the new configuration before allowing object mutations after configuration change. Reconfiguration protocol is used for moving the system from one configuration to the next.

The proposed system provides durability for all the transactions that are committed even if the entire cluster loses power or fails. System have five main phases of recovery after failure that includes failure detection, reconfiguration, transaction state recovery, bulk data recovery and allocator state recovery. In the later part author uses some experiments results to prove that

FaRM provides significantly higher throughput and lower latency than state of the art in-memory databases. FaRM can also recover from a machine failure back to providing peak throughput in less than 50 ms, making failures transparent to applications.

**Questions:**

**1 -** Isn’t it better to try to increases the efficiency of earlier implemented transaction system instead of introducing new one?

**2 -** How system handle the commit backup or commit primary failure?

**3 -** FaRM depend upon information of primary and backups of each data region. Isn’t this dependency a bottleneck for system?

**4 -** How the failure in any phase of commit protocol handled? Will the system start again from that point or completed phases of a single commit will be lost?