Sprint: High-Performance Data Management for Clustered Architectures
Lásaro Camargo†⋄‡, Rodrigo Schmidt†⋄, Fernando Pedone⋆, Marcin Wieloch**
†Ecole Polytechnique Federale de Lausanne (EPFL) ☎ Student ◦ Professor
‡Ecole Polytechnique Federale de Lausanne (EPFL)
Universidade Estadual de Campinas (Unicamp)
⋆Ecole Polytechnique Federale de Lausanne (EPFL)
Universidade Estadual de Campinas (Unicamp)
† Universidade Estadual de Campinas (Unicamp)
email: {first name}.{second name}@lu.unisi.ch

1 - Motivation
High-performance data management systems have traditionally relied upon specialized hardware and proprietary software, both invariably expensive. Even though powerful hardware infrastructures, built out of commodity components, have become affordable in recent years, software still remains an obstacle to open high-performance and high-availability data management systems. The Sprint project aims to overcome these obstacles.

The Sprint data management system is tailor-made for clusters of shared-nothing crash-recovery servers equipped with powerful processors, large main memories, and low latency networks. Sprint delivers high performance and availability by efficiently orchestrating in-memory database engines running in the individual servers, and by removing disk I/O from the critical path of transaction processing. In Sprint availability and performance should be achieved without sacrificing strong consistency.

2 - Architecture
Sprint revisits traditional architecture assumptions in the light of current hardware trends. It distinguishes between physical servers, part of the hardware infrastructure, and logical servers, the roles performed by the data management system. There are three types of logical servers: edge servers, data servers, and durability servers, which can be hosted alone by physical servers or with other logical servers.

Edge servers receive transactional requests from external clients and execute them against the data servers. Each data server runs a local in-memory database system (IMDB). IMDBs provide high transaction throughput and low response time by keeping all the data in main-memory and avoiding disk I/O. Applications using IMDBs are, in most cases, limited by the memory capacity of the server running the IMDB. Sprint partitions the database space into segments and stores them in several data servers, widening this limit to the aggregated memory capacity of the entire cluster. Data servers may also be replicated, allowing parallel execution of read operations and boosting performance.

Sprint uses the Paxos Commit protocol to terminate transactions; durability servers provide stable storage to the underlying consensus protocol used by Paxos Commit. Although durability servers do access disk, this access is sequential, resulting in very low latency. Moreover, this information is used by background processes building snapshots of the database for recovery purposes. Durability servers are replicated for high availability; the system is available as long as a majority of durability servers is operational.

Since transactions do not have to wait for data to be fetched from disk, fine-grain synchronization (e.g., row level) becomes less important in IMDBs. Therefore, several IMDBs synchronize transactions using coarse-grain mechanisms (e.g., table level). Sprint currently assumes that IMDBs synchronize transactions using multiple-readers single-writer concurrency control, a special case of two-phase locking. Executing update transactions serially has several advantages: First, serializability is guaranteed trivially. Second, the gain in performance from not having to deal with locks, or any other complex synchronization mechanism (e.g., timestamps), overrides the loss in concurrency. Third, deadlocks within a single IMDB cannot happen. Finally, although update transactions are serialized within a single IMDB, they can execute concurrently in different data servers.

Local transactions, those accessing a single data server, naturally have low overhead in terms of communication and are deadlock-free. Global transactions, those accessing data in multiple servers, on the other hand, will cost more in communication, processing, and deadlock detection. In Sprint, although coarse locking is used, global transaction do not impose prohibitive cost; primarily because of the low latency network and sequential disk access employed, and secondarily because Sprint orders global transactions, by means of a total-order broadcast, and ensures that their executions follow this order, efficiently avoiding deadlocks.

Summing up, Sprint combines high performance, high availability, and strong consistency by employing:

1 – Simple execution model. Local transactions have minimal overhead. Resolving distributed deadlocks requires neither timeouts nor complex data structures (e.g., wait-for graphs).

2 – In-memory transaction execution. Transactions are not limited by the memory capacity of a single data server. Transactions that would read from disk in on-disk databases access multiple data servers in Sprint.

3 – Low latency termination. Update transactions rely on Paxos Commit to manage atomic commitment. Paxos Commit generalizes two-phase commit (2PC) and has comparable latency without its blocking shortcoming.

4 – Decoupling of replication for performance from replication for fault tolerance. Data servers can use replication to allow parallel reads. Durability servers use replication to improve availability.

5 – Efficient recovery. Recovering a data server requires installing an image in some operational server, and applying new updates; it does not block the system.

3 - Future Steps
Sprint’s distributed data management protocol was designed to stress processors, main memories, and the network, while sparing disk. No restrictive assumptions are made about the failure model (e.g., one server must be always up). The execution model is very simple, favoring local transactions. An implicit assumption is that by carefully partitioning a database, many transactions can be made local, maximizing performance.

We have implemented a prototype of Sprint and are currently evaluating its performance under traditional benchmarks and workloads dominated by different mixes of read-only and update transactions, and local and complex transactions.