Dependable Eventual Consistency with Replicated Data Types

Résumé du projet de recherche (Langue 1)
Web applications rely on replicated databases to place data close to their users, and to tolerate failures. Many replication systems opt for eventual consistency, which offers excellent responsiveness and availability, but may expose applications to the complexity of concurrency and failures. To alleviate this problem, recent replication systems have begun to strengthen their interface with additional guarantees, such as causal consistency, which protects the application from ordering anomalies, and Replicated Data Types (RDTs), which encapsulate concurrent updates via high level object interface and ensure convergent semantics. However, dependable algorithms for these abstractions come at a high cost in metadata size. This thesis studies three related topics: (i) design of RDT implementations with minimized metadata; (ii) design of consistency algorithms with minimized metadata; and (iii) the limits of the design space. Our first contribution is a study of metadata complexity of RDTs. RDTs need metadata to provide rich semantics under concurrency and failures. Several existing implementations incur high overhead in storage space, in the number of updates, or polynomial in the number of replicas. Minimizing metadata is nontrivial without impacting their semantics or fault-tolerance. We design optimized set and register RDTs with metadata overhead reduced to the number of replicas. We also demonstrate metadata lower bounds for six RDTs, thereby proving optimality of four implementations. As a result, RDT designers and users can better navigate the design space. Our second contribution is the design of SwiftCloud, a replicated causally-consistent RDT object database for client-side applications, e.g., mobile or in-browser apps. We devise algorithms to support high numbers of client-side partial replicas backed by the cloud, in a fault-tolerant manner, with small and bounded metadata. We demonstrate how to support available and consistent reads and updates, at the expense of some slight data staleness; i.e., our approach trades freshness for scalability (small and bounded metadata, parallelism), and availability (ability to switch between data centers on failure). Our experiments with thousands of client replicas confirm the design goals were achieved at a low staleness cost.