Can Fault-Tolerant Disaggregated Cloud Applications Be Lightning Fast Too?

Tianyu Li
MIT CSAIL
litianyu@mit.edu

Flexibility is a key promise of the modern cloud. It is most easily achieved when developers build applications on managed cloud services, disaggregated across many auto-managed and auto-scaling components. Recent advances in microservices and serverless architectures will make the disaggregated application model even more prevalent and essential to business operations. That said, guarantees are difficult to come by in a disaggregated world, as disaggregated applications naturally distribute work across fine-grained components that can fail partially, frequently, and in unexpected and strange fashion (i.e. grey failures). The current state of the world today is such that cloud developers simply throw up their hands and embrace the lack of guarantees – at-least-once guarantees and idempotency is good enough for many workloads. However, it is not good enough for many other use cases, such as anything involving financial transactions (e.g., transfer of funds) or physical effects (e.g., rocket launches). Moreover, it is just simpler to write code with guarantees. Developers are more productive when not forced to reason about the complexity of duplicates and partial failures in distributed systems.

The key obstacle to stronger guarantees is latency. Recent attempts to build fault-tolerant layers on top of cloud services generally report hefty latency overhead of 10s, if not 100s of milliseconds. Unlike throughput, latency cannot be masked easily by dividing up work and throwing resources at the problem; latency adds up as developers compose applications from services; users are frustrated and hit refresh frantically, or worse, click the little red button on the top right corner. Designing and implementing newer or better services (e.g., stateful FaaS) does not address the problem where multiple services are composed to support disaggregated applications. We need a general way to mask latency in disaggregated cloud applications while retaining strong guarantees.

One might argue that this is not possible. Indeed, there are fundamental reasons why guarantees are expensive in the cloud – we incur cost whenever we replicate state, interact with persistent storage, communicate, or simply work on time-shared, jittery hardware. I argue instead that they can be *mitigated*. The key insight here is that guarantees need not apply in all the cases equally, due to the opaque nature of cloud applications. People only care about guarantees and hold the system accountable in select cases (e.g., that money doesn't appear if someone kicked the power cable during a transaction, that a rocket launch is properly authorized by relevant parties), and can tolerate rollbacks, retries, and repairs in between two such points, as long as the cloud transparently manages them. The key to taking advantage of this insight is *optimism*. Optimism means that we should achieve fault-tolerance assuming that failures are rare, and that therefore it pays off to proceed as if failure does not happen and repair only if failure occurs. In the context of disaggregated cloud applications, this means applications speculatively execute without waiting for guarantees, pausing to do so only before launching that rocket.

The complexity with this solution lies in the failure case – what if we made a bad bet? Optimistic schemes such as hardware speculative execution or optimistic concurrency control adopt a rollback-replay scheme. Doing so in the cloud requires us to carefully track dependencies between different components and restoring the application to a consistent past state, and faithfully replay application logic. It is challenging to efficiently track this many moving parts, and to handle nondeterminism or user input when replaying application logic; however, I believe it to be possible and have already worked out a scheme to do so in certain cases such as with Distributed Prefix Recovery (DPR), speculating on the persistence of entries in cache-storage shards, but there are many pieces. Can the rollback-replay model be incrementally integrated into the current, pessimistically fault-tolerant cloud services? Can the cloud runtime hide the complexity effectively enough from developers? Can we speculate on guarantees other than persistence, such as consensus or total ordering? I optimistically speculate that we can dramatically reduce the latency and increase the utility of fault-tolerant disaggregated cloud applications by applying these techniques. I will present our current toolbox of optimistic cloud services and lay out a road map to incrementally convert applications to our speculative execution models.