Consistency, Replication and Fault Tolerance

● Introduction to replication and consistency

There are two primary reasons for replicating data: reliability and performance. First, data is replicated to increase the reliability of a system. If a file system has been replicated it may be possible to continue working after one replica crashes by simply switching to one of the other replicas. Also, by maintaining multiple copies, it becomes possible to provide better protection against corrupted data. For example, imagine there are three copies of a file and every read and write operation is performed on each copy. We can safeguard ourselves against a single, failing write operation, by considering the value that is returned by at least two copies as being the correct one.

The other reason for replicating data is performance. Replication for performance is important when the distributed system needs to scale in numbers and geographical area. Scaling in numbers. occurs, for example, when an increasing number of processes needs to access data that are managed by a single server. In that case, performance can be improved by replicating the server and subsequently dividing the work.

Scaling with respect to the size of a geographical area may also require replication. The basic idea is that by placing a copy of data in the proximity of the process using them, the time to access the data decreases. As a consequence, the performance as perceived by that process increases. This example also illustrates that the benefits of replication for performance may be hard to evaluate.

The problem with replication is that having multiple copies may lead to consistency problems. Whenever a copy is modified, that copy becomes different from the rest. Consequently, modifications have to be carried out on all copies to ensure consistency. Exactly when and how those modifications need to be carried out determines the price of replication.

Replication and caching for performance are widely applied as scaling techniques. Scalability issues generally appear in the form of performance problems.

Placing copies of data close to the processes using them can improve performance through reduction of access time and thus solve scalability problems. A more serious problem, however, is that keeping multiple copies consistent may itself be subject to serious scalability problems. Intuitively, a collection of copies is consistent when the copies are always the same.

This means that a read operation performed at any copy will always return the same result. Consequently, when an update operation is performed on one copy, the update should be propagated to all copies before a subsequent operation takes place, no matter at which copy that operation is initiated or performed.

This type of consistency is sometimes informally (and imprecisely) referred to as tight consistency as provided by what is also called synchronous replication. The key idea is that an update is performed at all copies as a single atomic operation, or transaction. Unfortunately, implementing atomicity involving a large number of replicas that may be widely dispersed across a large-scale network is inherently difficult when operations are also required to complete quickly.

Difficulties come from the fact that we need to synchronize all replicas. In essence, this means that all replicas first need to reach agreement on when exactly an update is to be performed locally. In many cases, the only real solution is to loosen the consistency constraints.

In other words, if we can relax the requirement that updates need to be executed as atomic operations, we may be able to avoid (instantaneous) global synchronizations, and may thus gain performance.