CMPSCI 677 Operating Systems **Spring 2013**

Lecture 16: March 25

Lecturer: Prashant Shenoy *Scribe: Srikar Damaraju*

16.1 Review

16.1.1 Timestamp Based Concurrency Control

Timestamp-Based Concurrency Control is based on Lamport's clocks. Each transaction Ti is a given a timestamp ts(Ti). If Ti intends to do any operation that conflicts with another transaction Tj with timestamp ts(Tj), then ts(Ti) is compared with ts(Tj). If ts(Ti) <ts(Tj), then transation Ti is aborted, and then later restarted with a larger timestamp.

The idea here is that a transaction with a larger timestamp occurs later in time than a transaction with lower-valued timestamp. Since, transaction with lower-valued timestamp is preceding in time, it is aborted.

16.1.2 Types of Replication

Replication can be of two types

Data Replication Multiple replicas (copies) of data are stored on multiple machines. All replicas represent the same data

Computation Replication Same application runs on multiple machines

16.1.3 Reasons for Replication

Reliability If a machine holding the data crashes, then the whole system will not have access to the data. Having replicas of data will ensure high availability of data

Performance Replication can be used to scale a distributed system, and serve large number of requests. Geo-Replication gives better latencies, by serving clients using closest replica to those clients.

16.1.4 Replication Issues

Typically issues concerning replication are application specific. It is up to the application designer to choose what kind of replication is needed. One of the major issues of replication is Consistency. The system should guarantee that all replicas are consistent and represent same data. There are notions in consistency known as Strong Consistency, and Weak Consistency, based on the guarantee of consistency given to the user.

16.1.5 Design Choices

Approach 1 Application will be responsible for replication, and consistency. This approach offers flexibility for application designers to choose their own mechanisms. In this case, designer will be responsible for consistency, which is an added burden for designer.

Approach 2 System (middleware) takes care or replication, and consistency.

16.1.6 Replication and Scaling

Replication and Caching are typically used for scaling systems. Consistency mechanisms are chosen based on the frequency of Read/Write operations on replicated data. Say for a system with N copies of data, 'R' read frequency, and 'W' write frequency. If R¿¿W we may choose a stronger form of consistency policy saying that every Write is immediately reflected on all other replicas. If W¿¿R, we may choose a policy saying that "main" copy is updated for every Write, and when a Read comes in, all other replicas will be updated. This will reduce the overhead of sending updated messages often in a system with W¿¿R. These consistency policies depend upon applications and their requirements.

16.2 Consistency Semantics

16.2.1 Strict Consistency

Strict Consistency policy assumes that all machines in the system have global clock, and imposes that every Write operation is reflected immediately on every other replica in the system.

16.2.2 Sequential Consistency

This is a looser form of consistency compared to Strict Consistency. Sequential Consistency assumes that all operations are executed in a sequential order. All processes agree on same interleaving, while preserving their own program order. There is no global clock in this policy.

16.2.3 Linearizability

Linearizability is stronger than Sequential and weaker than Strict Consistencies. It assumes sequential consistency. This policy says that if two processes exchange a message, then they should respect the order of events, otherwise they can agree upon any order.

16.2.4 Causal Consistency

This policy says that Causally related writes must be seen by all processes in the same order, but concurrent writes may be seen in different orders on different machines.

16.2.5 Other Models

FIFO Consistency Writes from a process are seen by others in the same order. Writes from different processes may be seen in different order (even if causally related). This relaxes causal consistencies. FIFO consistency requires that writes from a process should be seen in the same order.

Using Granularity of Critical Sections This policy uses granularity of critical section, instead of individual read/write. It only considers ordering before entering critical section and after completing the work in critical section.

Entry and Release Consistency This policy assumes that shared data are made consistent at entry or exit points of critical sections

16.2.6 Eventual Consistency

Eventual consistency says that eventually over time all replicas will see all updates and reach a consistent state. This does not guarantee any consistency while the updates are being made. It is a very weak consistency policy.

Examples: DNS, NIS

16.2.7 Epidemic Protocols

Epidemic protocols update replicas based on theory of epidemics. It says upon a new update try and "infect" other replicas as quickly as possible. It will be discussed in-detail in the next lecture.