Distributed Data Storage and Management Part VII: Ensuring availability

Saptarshi Pyne
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Jodhpur, Rajasthan, India 342030

CSL4030 Data Engineering Lectures 19, 20, 21
September 20th, 22nd, 27th, 2023

What we discussed in the last class

- Locking protocols for concurrency control in a distributed database
  - Single lock manager protocol
  - Distributed lock manager protocol
  - Majority protocol (with ordered lock acquisition for avoiding global deadlocks)
  - Biased protocol
  - Quorum consensus protocol

Remaining sub-topics for distributed databases

- Availability
  - High availability at the cost of consistency: The Cloud
- Multi-database systems for heterogeneous distributed databases
- Distributed directory systems for managing data
  - The lightweight directory access protocol (LDAP)

Availability of a distributed database

- Detect and Recover strategy
  - Detect a failure and type of the failure
  - Recover from the failure
- How to distinguish a link failure from a network partition?
  - If we can find an alternative route to the target site, it is a link failure.
- How to distinguish a site failure from a network partition?
  - Very difficult to distinguish them.

Distributed locking protocols for high availability

- Read one, write all ‘available’ protocol
  - Demerit: ‘Site reintegration’ (integrate relevant changes made to the database while a site was down) is difficult

Distributed locking protocols for high availability (contd.)

- Majority-based protocol
  - Assign a version number (an integer) to each replica
  - A read/write lock must be obtained on ceil(n/2)+1 replicas
  - For read(Q), find the replicas with the highest version number and read any one of them
  - For write(Q), first do read(Q) as mentioned above. Then update all locked replicas (including the ones with lower version numbers). Finally, update the version number of all locked replicas to new_ver = (prev_highest_ver + 1)

  - Merit: Site reintegration is trivial

Coordinator failure handling strategy for high availability

- A backup coordinator
  - When the primary coordinator is up, any changes to the primary coordinator files (such as the lock table, incoming message log, outgoing message log) are immediately backed up in the backup coordinator.
  - As soon as a coordinator failure is detected, the backup coordinator goes live.

  - Demerit: Network partitions

Coordinator failure and network partition handling strategy for high availability: The bully algorithm

Assign a unique int (ID number) to each site = S_i

The site with the highest ID is selected as the coordinator.

For each site S with ID number S_i

If (no messages received from the coordinator in a certain time period)

Ping all sites with ID>S_i and start timer(T);

If (no ping back from any higher ID site)

Message all the lower ID sites that now it is the coordinator i.e. **bully** all the lower ID sites;

Else

Start timer(T') and wait for a message from the new coordinator;

If (no message received)

Message all the lower ID sites that now it is the coordinator;

Coordinator failure and network partition handling strategy for high availability: The bully algorithm (contd.)

When there are network partitions, each partition will have a bully coordinator.

Merit: It ensures that there is always **exactly one coordinator** in charge of **a partition**. When the whole network is integrated again, we only need to ensure reintegration of the bully coordinators which is less cumbersome than reintegrating each individual site.

When should we prioritize availability over consistency?

Consistency is critical for some apps (e.g., banking apps) but not so much for other apps (e.g., social network apps). Consistency of a bank account balance is critical across sites. However, temporary inconsistency in the number of likes on a post across sites is acceptable.

The CAP theorem:

A distributed database can have **at most** two of the three following properties:

- Consistency
- Availability
- Partition-tolerance

Proof (out of syllabus): Gilbert, Seth, and Nancy Lynch. 'Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services'. ACM SIGACT News 33.2 (2002): 51-59.

Explanation of the CAP theorem

Consistency: An execution of **a set of operations** (reads and writes) on replicas is said to be **consistent** if its result is the same as if the operations were executed at a single site in the order they were issued by the parent **transactions**.

It refers to consistency of a data item after executing a set of operations which can consist of **multiple transactions**.

If consistency is critical for an app (such as a banking app, a online multiplayer gaming app), we need to honor the ACID properties.

If inconsistency can be temporarily accepted at the benefit of availability and partition-tolerance (e.g., Instagram), we need to honor the **BASE properties**.

The BASE properties

Basically **A**vailable:- If a network partition occurs, the replicas in a particular partition can be read and written independently.

Soft state:- The state of a replica should be consistent with that of the other replicas in the same partition. However, it might be temporarily inconsistent with the replicas in other partitions.

Eventually consistent:- Once the network partition is over, all replicas across all partitions should become consistent. Restoring consistency is usually handled by the app itself, not the DBMS unless the DBMS is custom made for that app, e.g., Tao is custom made for Facebook.

Why does the app itself handle the task of consistency restoration?

Suppose, before a network partition, a photo on Instagram had 15 likes. Then a network partition occurred, creating two partitions. During the partition, 5 new users liked the post in each partition.

What should be the total number of likes on the post after the network partition is over?

References

- A. SILBERSCHATZ, H.F. KORTH, S. SUDARSHAN (2011), Database System Concepts, McGraw Hill Publications, 6th Edition.
  - Chapter 19. Distributed Databases

- Paper: Bronson et al., “TAO: Facebook’s Distributed Data Store for the Social Graph”, 2013 USENIX Annual Technical Conference (USENIX ATC ‘13).
  - Video: https://www.usenix.org/conference/atc13/technical-sessions/presentation/bronson

Thank you