Data –centric consistency 는 Tight 한걸 loose하게 하였다.

-> sequential, causal, fifo consistency

But Sequential 이든 causal이든 fifo든 결국 이 시스템을 구성하는 서버들끼리 계속해서 consistency를 유지하기 위해서 뭔가를 계속 해야 한다. 이 또한 코스트가 커서 Eventual consistenc

Eventual consistency

-> 일정 기간 후에 consistency를 유지시켜주겠다.

But 모바일 시대가 오면서 스마트폰 유저들의 이동 때문에 consistency유지가 힘들어 졌다.

4가지

1. monotonic reads

2. monotonic writes

3. read your writes

4. write follows reads

위의 4가지 덕분에 eventual consistency 만으로도 client에게는 consistency 한 환경을 제공해 줄 수 있게 되었다.

=> client –centric consistency(model)

Notations:

W( x1 ; x2 )

세미콜론이 있으면 order : x1 -> x2

x2가 x1를 알고있는 상태에서 x2를 생성했다. (causal 하다)

W( x1 | x2 ) -> Concurrent 하다.

Consistency Protocols

- Data – centric consistency ( sequential consistency를 구현한다.)

Primary-backup protocols

Local-write protocols

Quorum-based protocols

Primary-backup protocol -> centralized algorithm

Quorum-based protocols -> decentralized algorithm

1. Primary – backup protocols

All write operations need to be forwarded to a fixed single server(=primary server)

Primary - backup protocol은 한 클라이언트가 write하면 그게 primary server로 가서 모두에게 전파될 때까지 다른 클라이언트가 read할 수 없게 blocking 하는거고

2. nonblocking approach는 한 클라이언트가 write하면 그게 primary server에 write되는 순간 ack을 리턴해서 read가 가능하게 하는 것

* Blocking vs. non-blocking?
  + Write speed? //Nonblocking이 더 빠르다
  + What happens if there is an error after W3? Consistency problem?

nope

* + Sequential consistency? (Does all servers see the same sequence of writes?)

Of course

* + Tight consistency? (Can client always see the most recent copy?)

Only in blocking

3. Local – write protocol

처음에 write request가 들어오면 기존 primary server가 그 서버로 item을 보내주고 primary server가 바뀌게 된다. 그리고 client에게 write가 끝났다고 reponse를 바로 해주게 된다. 그 이후에 update 된 내용을 전파한다. 그래서 request를 하고 respond 받을 때까지 기다릴 필요가 없다..

4. Quorum – based protocols(tight consistency도 만족)

* File is replicated on **N** servers. Need **permission** from **multiple replica servers** for file read (**NR**) and write (**NW**).

• Writes are executed at a subset of all replicas, called a write quorum

• To read a data item, the process must contact a subset of replicas, called a read quorum, to find out the newest version of the data item and read it

**NR** + **NW** > N (prevents read‐write conflicts)

**NW** > N/2 (prevents write‐write conflicts)

- Client-centric consistency(Implementing monotonic-read cinsistency)

내가 그 동안 뭘 읽고 뭘 썼는지 set을 가지고 있다. 내가 process 1에서 어떤 값을 만들어냈다고 하자. 그럼 ID 가 생성이 되고 E.g. Lamport time stamp + processor number

내가 write를 하면 write set 에 들어가게 되고 내가 뭔가를 read해도 이 또한 뭐가 썼던 거기 때문에 write ID가 read set에 들어간다. 쉽게 말해 내가 어디서 뭘 읽고 뭘 썼는지 기록이 남아있다.

클라이언트가 server에서 read를 하려고 할 때 클라이언트의 read-set을 확인해서 모든 write가 로컬에서 수행되었는지 확인한다.

만약 아니면 해당 로그 정보를 사용하여 다른 서버에 연결하여 missing writes를 가져오고 수행한다.

또는 최신 서버로 read를 forward한다.

After performing the read, the read-set is updated

* + The algorithm requires that
    - The write-id contains the server id who made the write
    - The order of the write operations is also recorded (LamportTS)