MongoDB教育訓練-20211223-02

- 資料先寫入記憶體 ，
- checkpoint => 一分鐘(or 資料到達大小) 髒數據寫入DB
- journal log 60ms 寫入 db
- db 最多錯失60 ms 資料
- "OK - committed" => 寫到哪裡？ write concerns 決定
- 資料遺失的時候？我會知曉？
- knowingly log => 已知的寫入失敗
- unknowingly log => 未知的寫入失敗

Write Concerns - Questions

When you write to the database:

What does Durable mean to you?

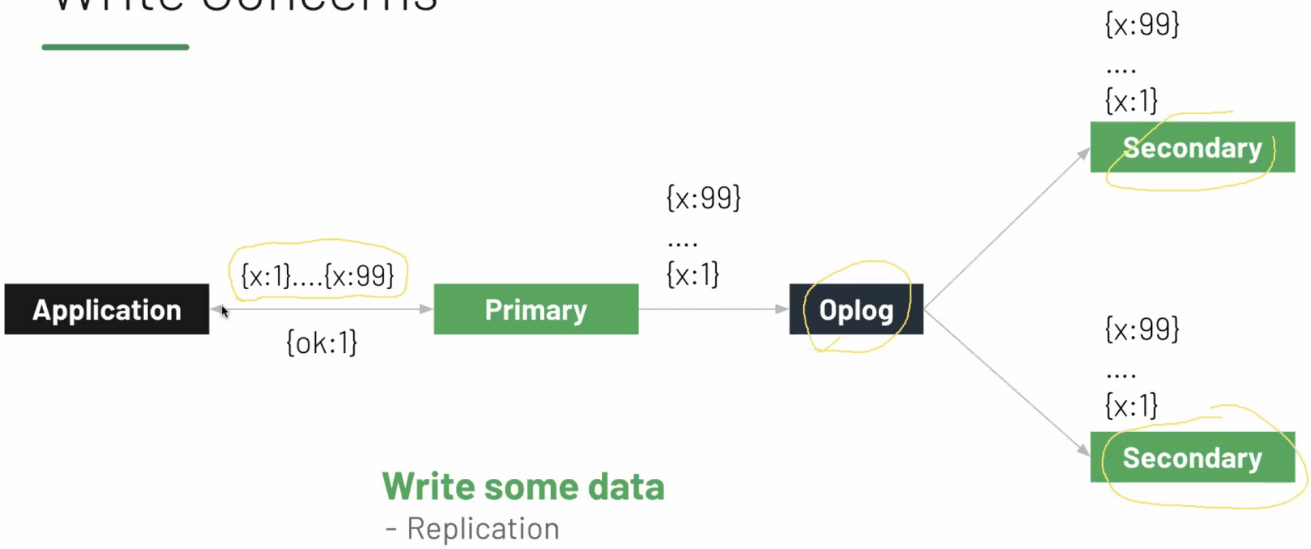
What does 'OK - committed' from the database mean?

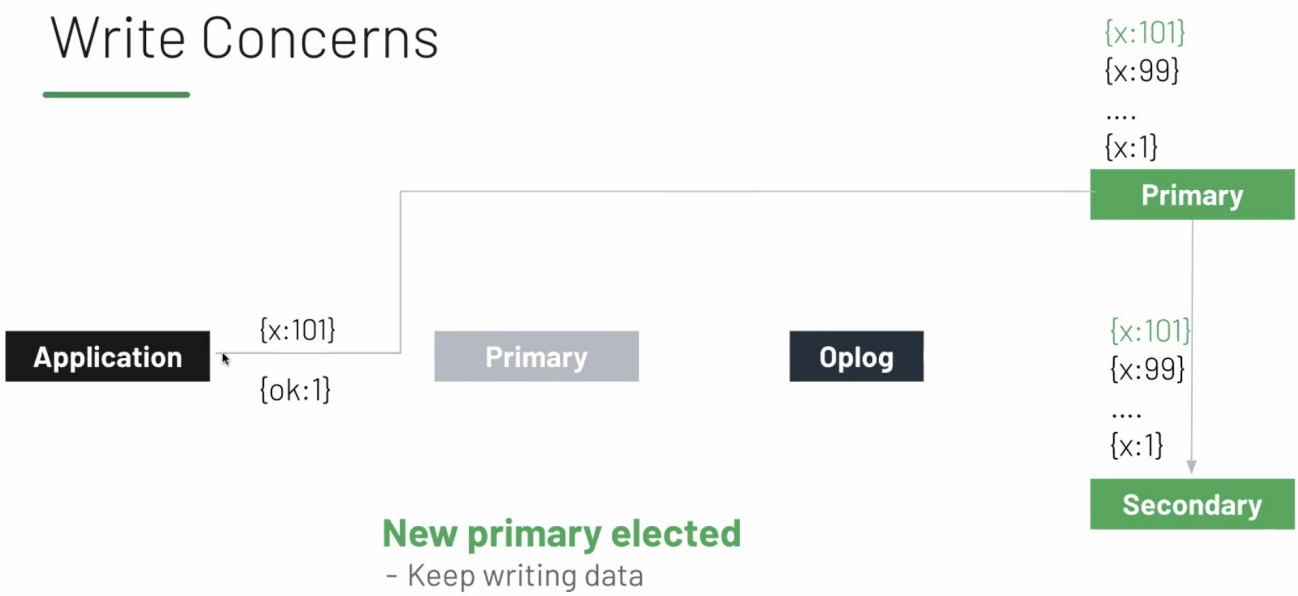
What sort of data would not matter if the latest was lost in a crash?

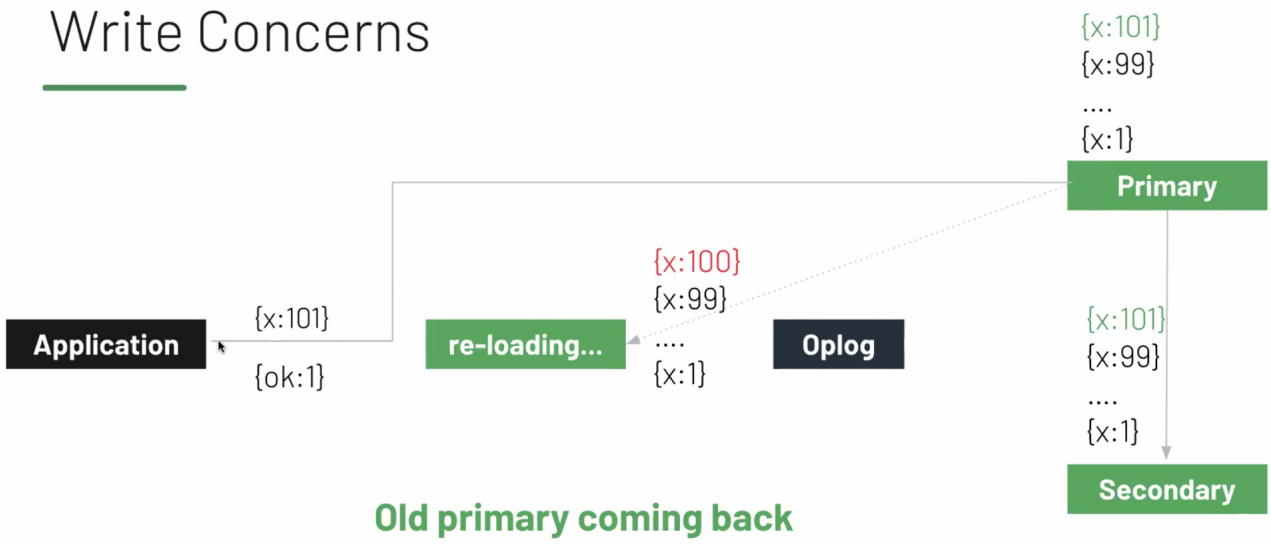
Who decides what data must never be lost?

Is there a difference between knowingly lost and unknowingly lost?

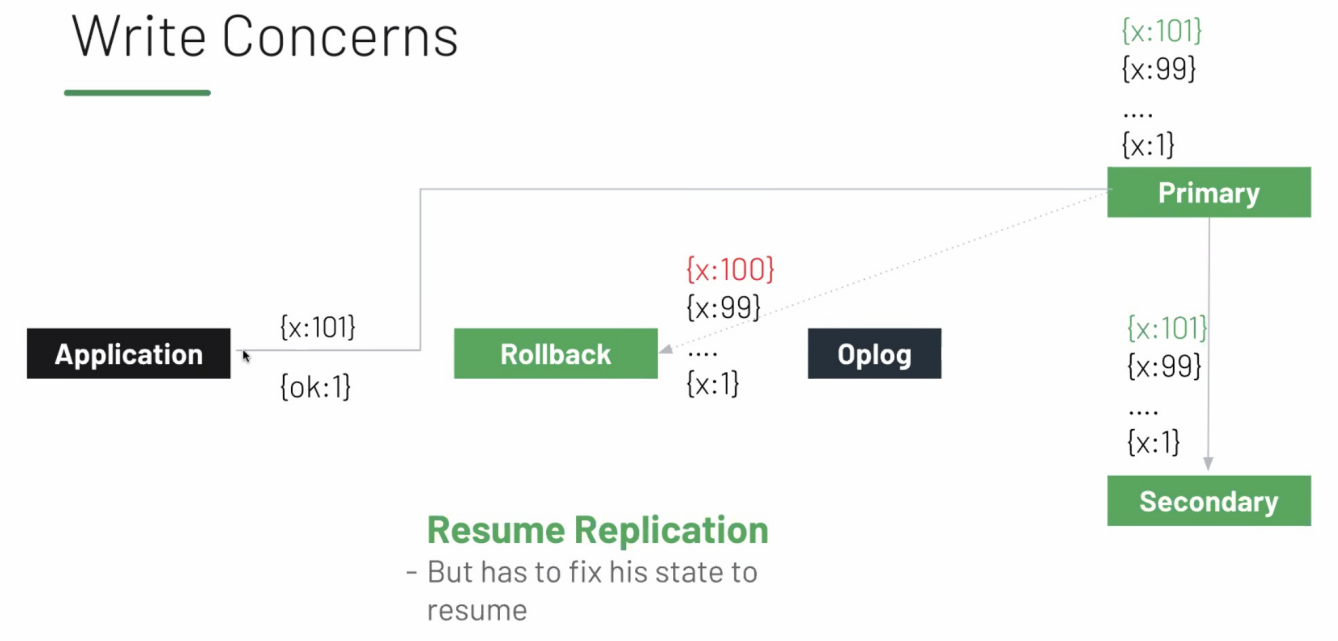
Write Concerns

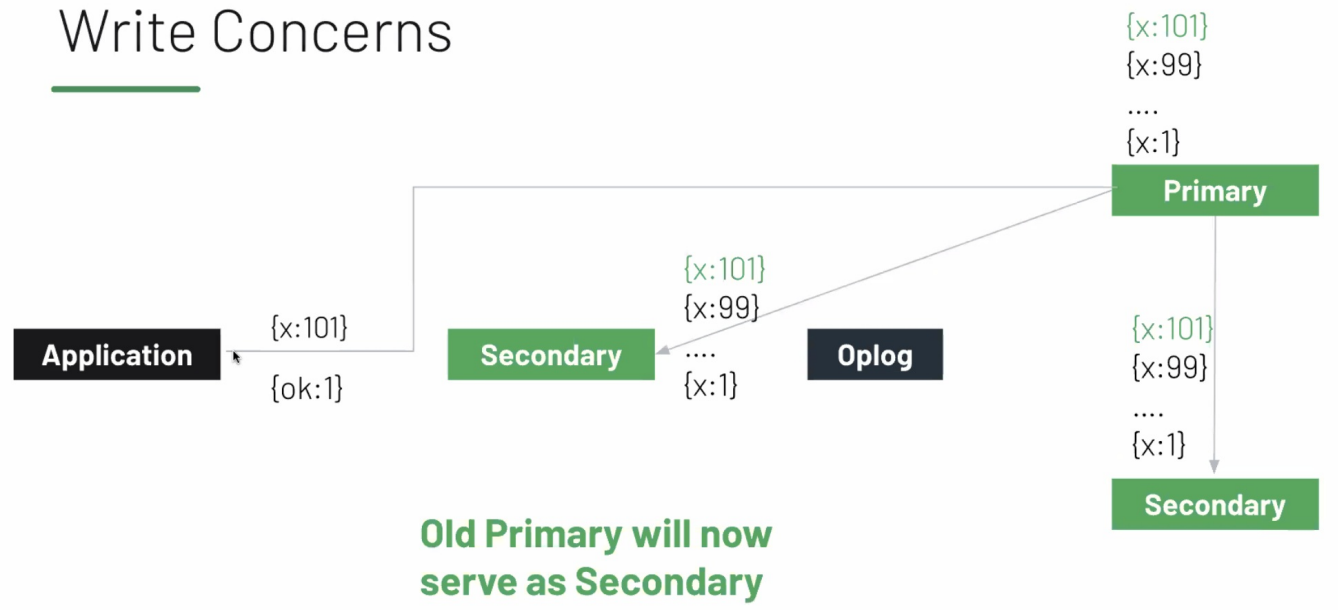






當資訊不同步，server 必須 rollback 確保資訊一致 => { x: 100} (移除)





The case for Majority writes

in the previeous scenario:

date is writtten to the primary

primary acknowledges write to application

primary dies before secondary reads data

必須所以資料到達大多數節點(majoruty commit point)，才能稱之為持久化

The case for Majority Writes

In the previous scenario:

Data is written to the primary.

Primary Acknowledges write to application.

Primary dies before secondary reads data.

Secondaries have an election.

All trace of the acknowledged write (x=100) was silently lost!

The Majority Commit Point

An important concept to understand the upcoming slides:

- The Primary knows what timestamp each secondary is asking for.
- It, therefore, knows they have everything before that durable.
- It, therefore, knows up to what timestamp exists on a majority of nodes.
- And consequently, up to what point data is 100% safe.
- Until a change is 100% safe, the Primary will keep it in memory.

Note that in a system with automated failover - safe means "If it fails over seamlessly, this won't be silently lost."

寫的關注度Write Concern

- OK, commitresd 意義
- (w: 到達多少個節點)(j: 如何才算寫成功 0:記憶體, 1:硬碟)
- w:0 不推薦
- w:1, j:1
- w: "majority" (j: 1)
- w:3 無意義(假設叢集有3節點，萬一1節點丟失，無法滿足)

Write Concerns

- MongoDB lets you specify what 'OK, committed' means.
  - Received by the primary over the network but not examined. (w :0)
  - Received and written by the primary - durable on primaries disk. (w :1, j : 1)
  - Received and written by a majority. (w : "majority")

w is the number of servers, *j* is whether to wait for the next disk flush (default with majority)

You can specify these in your application on any write, or on a connection, or an object you use to write.

MongoDB will wait until it achieves the level you request or times out. If it times out, it may still have done some part of it. In the event of a timeout, you may need to confirm the state.

Why not just always write to a majority?

```
var wc = { w: 0 }
totalrecs = 10000
batchsize = 1
nbatches = totalrecs / batchsize
var start = new Date()
for(x=0;x<nbatches;x++) {
 recs = []
 for(y=0;y<batchsize;y++) {recs.push({a:1,b:"hello"})}
 db.test.insertMany(recs,{writeConcern: wc})
}

var end=new Date()
print(`${end-start} milliseconds`)
```

Exercise - Write Concerns

The Manual page for insertOne shows we can specify the write concern using

```
db.collection.insertOne( <document>, {
      writeConcern: { w: <value>, j: <boolean>, wtimeout: <number> }
})
```

In the shell, run the supplied code to measure the time taken to insert 10,000 small documents using different write concerns and values of j. Use a for loop. Complete this table

	Durability Guarantee			
Batch size	w:0	w:1	w:1, j:true	w:"majority"
1				
100				

How would the relative location of the client and servers impact this?

讀的關注度 Read Concerns

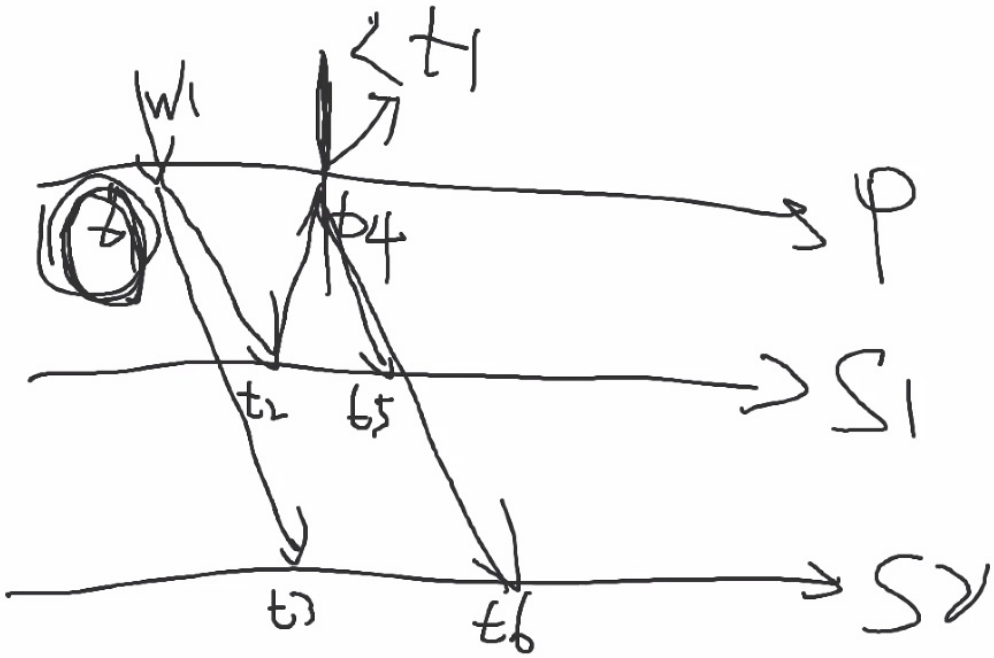
RDBS 隔離等級概念

Read local 主節點損毀才是髒讀

Read majority 讀舊一點的資料

Read snapshot :

Read Linearizable: 讀取時會確認資料是否到達大多數節點



Read Concerns

When reading you can choose how reads are impacted by what's durable

Read Local	What's the latest on the Primary
Read Majority	**What's the latest that is 100% durable** Enabled by Majority commit point.
Read Snapshot	**Read what was there when our query starts.** This hides any changes whilst the query is going on. But we need to keep data around whilst we do it.
Read Linearizable	Wait until a majority catch up with my query time

讀取資料會從哪個節點讀？

- Read from nearest Geographically (從最近的節點 ping 值，最小的)
- Read from a specific set of servers (設定標記好的 tag server)

Read Preferences

When do you think you would use each of the following?

- Read from Primary Only
- Read from Primary unless no Primary exists (primaryPreferred)
- Read from any Secondary Only
- Read from Secondary unless no secondary exists.
- Read from nearest Geographically
- Read from a specific set of servers

Which read preference would you use and why?

非必要不使用

Arbiter

- An Arbiter does not have a copy of the data set and cannot become a primary.
- An Arbiter participates in elections for primary and acts as a tie-breaker.
- Arbiters are strongly advised against in production systems.
- A system with Arbiters can be Highly Available OR Guarantee Durability
  - But not both

Recap

- Replication creates multiple identical copies of data.
- Typically this is used for High Availability
- Writes are to a Primary and replicated to Secondaries
- The primary is elected by the cluster as needed.
- This replication happens via the oplog
- A write concern of "majority" is the one that ensures data durability.
- The default write concern is 1 for a Replica Set ("majority" in MongoDB 5.0+)
- Friends don't let friends use Arbiters.

⏲修訂版本 #5
★由 treeman 建立於 23 ⍰Q⍰G⍰⍰ 2021 21:36:26
✎由 treeman 更新於 5 ⍰⍰⍰⍰ 2023 10:14:59