Review
Tunable Consistency in MongoDB

Auer Thomas

November 9, 2022

The paper ”Tunable Consistency in MongoDB” reviews consistency trade-offs when using the NoSQL database management system MongoDB. The document oriented database provides a variety of consistency levels that the client applications can utilize at operation level. Many applications may allow infrequent periods of consistency given the high cost of ensuring consistency at all times, at all layers of the DBMS. The PACELC theorem, an extension of the CAP theorem, is used as guideline as to what DBMS characteristics are important for the specific use-case of a given environment.

To further provide tunable consistency options, MongoDB implements the parameters for ’WriteConcern’ and ’ReadConcern’. WriteConcern specifies to which point the data must be commited to the whole, sharded system until the commit-success is passed on to the client. As such, the parametrization of ”w:1” will acknowledge the commit as soon as the write is commited on the primary node that serviced the write, whereas ”w:majority” will require the data to be commited to a majority of all secondary, sharded nodes.
ReadConcerns works similarly, in which the data returned must be commited to a certain number of nodes. ”r:1” will read the data directly from the primary node, whereas ”r:majority” requires the data that is requested to be commited on a majority of nodes.
With the correct parameter setup, namely ”r:linearizable” and ”w:majority”, MongoDB is able to provide the strongest consistency guarantees. It is noteworthy that MongoDB users tend to use ”w:1” and ”r:1” (as default) based on the importance of latency rather than consistency. Yet the decision to use weaker consistency levels often works as failures are infrequent and data losses are managable small.

Since MongoDB allows multiple consistency levels in the same deployment, it is important that this does not impact the performance of the whole system.
Writes demand high latency as the data is written to the primary and subsequent secondary nodes. Reads are more complex, given the WiredTiger engine in the backend – Reads are performed with snapshot isolation, meaning that all later updates must be kept in memory. Time demanding reads, hence, cause memory pressure which must be avoided to limit the impact on transaction performance. This is achieved by ”Query Yielding”, where the data is locked for regular intervals. After such an interval ends, the transaction aborts and releases the locks.

Table 4: Cross-AZ Latency Comparison of Write Concern Values in Milliseconds

	Client 1		Client 2	
Write Concern	Avg	99%	Avg	99%
`{w:1}`	1.83	2.92	2.01	3.02
`{w:"majority"}`	4.85	5.95	4.32	5.25

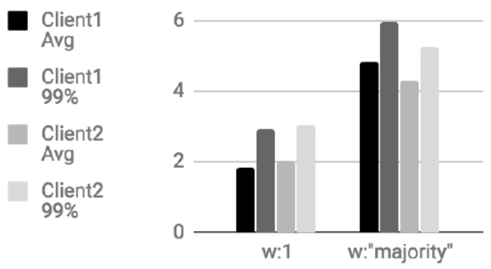

Figure 5: Cross-AZ Latency Comparison of Write Concern Values in Milliseconds

Table 5: Cross-Region Latency Comparison of Write Concern Values in Milliseconds

	Client 1		Client 2	
Write Concern	Avg	99%	Avg	99%
`{w:1}`	2.16	14.9	72	73
`{w:"majority"}`	185	192	217	480

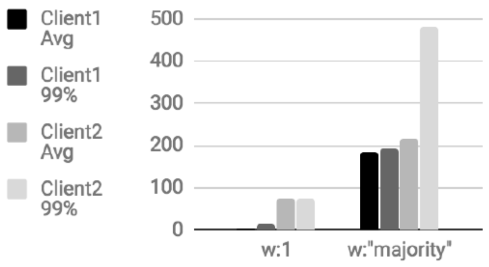

Figure 6: Cross-Region Latency Comparison of Write Concern Values in Milliseconds