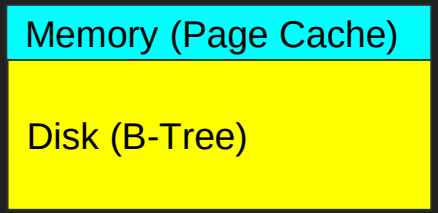
Aurora

Background

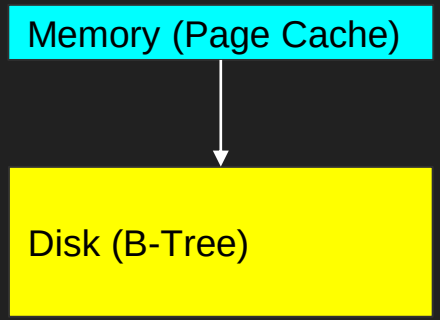
AWS Aurora is another NewSQL database which claims to gain a 35x performance boost over running traditional MySQL in the cloud. It uses some interesting concepts that involve much of what we have spoken about in the past, such as quorum writes for replicated logs as well as using caching in a clever way.

Traditional RDBMS Design



Traditionally, pages from the disk that need to be read or modified are first loaded from disk into memory. If the page is modified in memory, first an entry is appended to the write ahead log and then the changes in memory are eventually written back to disk (does not necessarily have to be done right away as long as the write is in the log).

Aurora Design

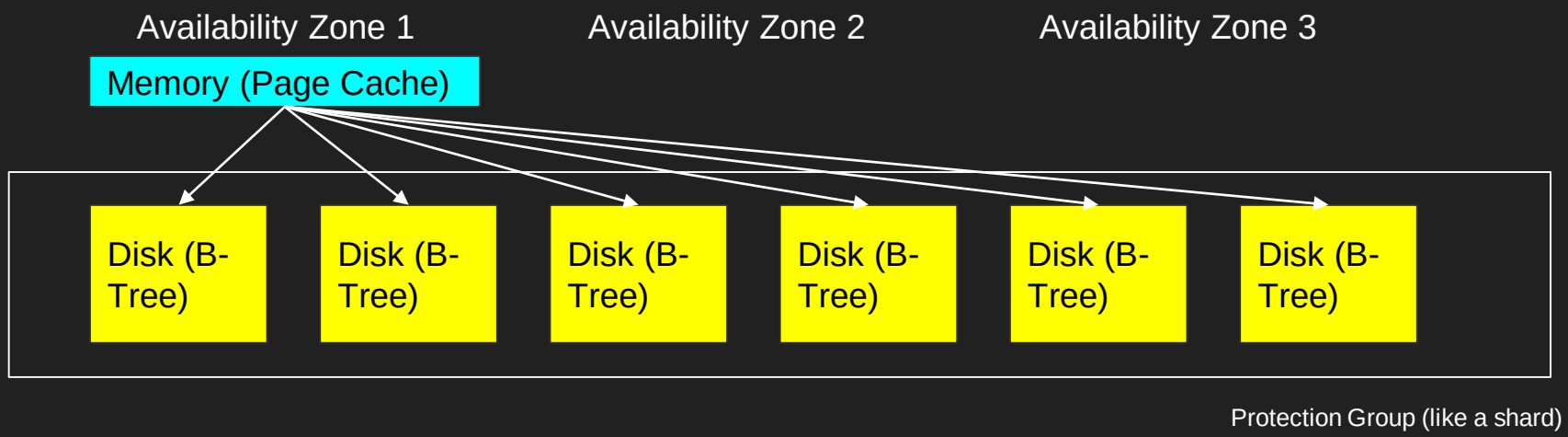


Aurora decouples the cache and disk on two different servers!

Takes too long to send entire modified page over network to disk (typically 8kb), instead just send new write ahead log entries to it, let disk server derive state locally.

Both disk and memory can be replicated independently!

Replication of Storage



Writes

Require a quorum of nodes in order to be successful!

- In this case, it is 4 out of 6 nodes
- Ensures high availability if an entire availability zone goes down

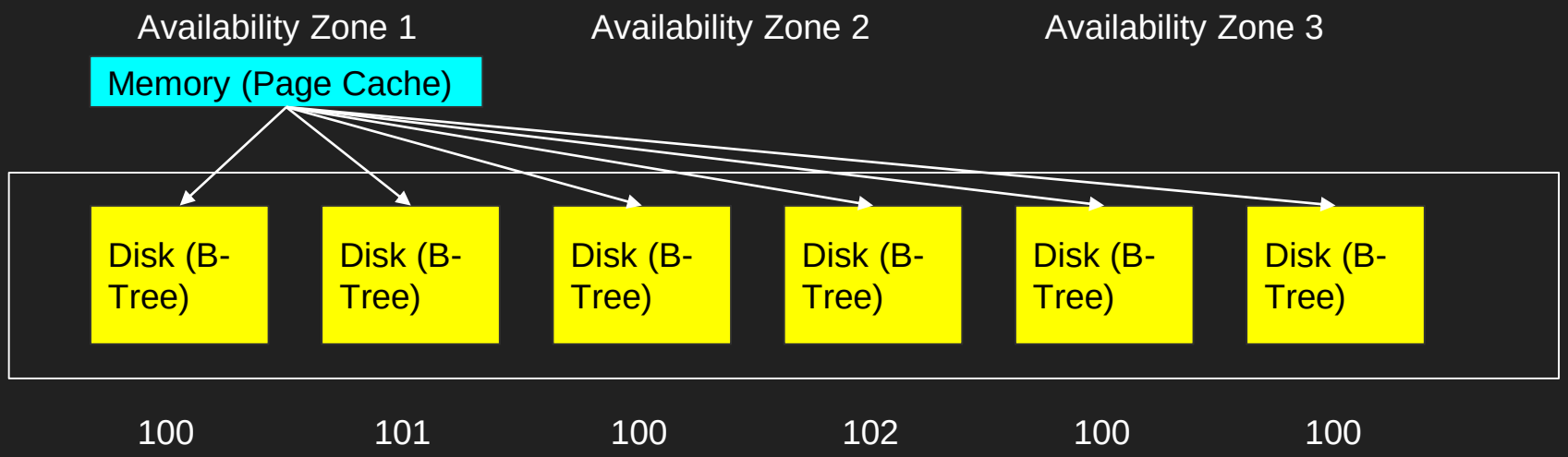
Reads

Reads do not have to use a quorum! The caching layer (database node) can simply look at its log to see what the log position is at the time the read was issued, and then only issue a log from an up to date replica.

Replication of Caching Layer

Can also add some read only cache replicas, main leader cache replica asynchronously propagates log changes to each of them. Each read-only replica can issue eventually consistent reads using its log in the same way that the main replica would (but since log is itself being propagated asynchronously reads can be stale).

Failure of Database Node



If a new node needs to take over as leader database node, it needs to ensure that any uncommitted writes issued by the last leader are reverted. It uses a quorum read to figure out the greatest committed log position (pictured above, 100), and reverts all log entries with a greater index on all of the replicas.

Failure of a Storage Node

If a server holding 10gb storage partition fails, it is likely that it was holding hundreds or perhaps even thousands of these partitions. In order to avoid massive network overhead of copying each partition to one other storage node, the shards on that storage node are sent to a variety of different storage nodes to parallelize the process of copying data over the network.

Conclusion

AWS Aurora is yet another example of clever engineering in order to help scale out the traditional relational data model. It acts as an interesting application of decoupling storage and caching, while also maintaining high availability and strong consistency (in some cases).

While it is not overly necessary to know all of the intricacies of Aurora, being able to mention it in a SQL vs. NoSQL interview discussion could be very useful, as it falls under the category of NewSQL.