

MySQL Sharding Business Challenges

📌 Standard / 👤 by Dave Anselmi (<https://www.clustrix.com/author/dave-anselmi/>) / 📅 June 29, 2017 / 💬
No Comments (<https://www.clustrix.com/bettersql/mysql-sharding-business-challenges/#respond>)



MySQL sharding comes with challenges. Previously we discussed **application and design challenges** (<https://www.clustrix.com/bettersql/challenges-sharding-mysql/>) that MySQL sharding requires. Today we'll expand on the MySQL sharding business challenges that can result and affect your business flexibility.

MySQL applications seeking scale often consider sharding. Sharding is a database architecture strategy in which the data workload is partitioned across multiple separate MySQL RDBMS servers. This kind of partitioning across separate servers allows MySQL applications to scale out writes as well as reads. And that's not trivial; write scale isn't something that read slaves or master-master solutions can provide to MySQL applications seeking scale. Read on for details about the MySQL sharding business challenges.

MySQL Sharding Business Challenges, #1: Business Flexibility

Previously we mentioned the issues MySQL sharding creates with cross-node transactions and **ACID transactionality** (<https://www.clustrix.com/bettersql/acid-compliance-means-care/>). Specifically, cross-node transactions aren't supported natively by a sharded MySQL array, because no single MySQL instance can manage multiple MySQL shards.

But what does that mean for business flexibility? Business rules supported by the MySQL sharded application resolve two main technical questions:

1. Does the application need to access data from multiple shards in a single database transaction (i.e., “cross-node transaction”)?
2. Do the application transactions need **ACID compliance** (<https://www.clustrix.com/acid-database/>)?

To shard effectively, data is partitioned across ranges. If all the data the application needs to access resides on the same shard, then everything is hunky dory. That transaction will behave just as if the workload is deployed on a single MySQL server, because, effectively, it is for that specific transaction. No cross-node data access or coordination is needed.

However, if the data you need to access is outside that range, that transaction will need to leverage a cross-node transaction.

What kinds of business rules need cross-node transactions?

- **Real-time lookups:** For example, if the full product catalog isn't available on each shard, then different ranges of users will be restricted in what they can access. That means the full product catalog would need to be replicated to each shard, and any changes to that catalog will have some level of latency from whichever shard is catalog master.
- **Upselling or cross-selling:** (i.e., “Who else bought that item” and “People buying that item also bought this other item(s)”: Behavior of users not co-located on the same shard won't be available for reference. Correspondingly, if all user behavior is extracted from all the shards, the resulting analytics still needs to be available to each shard for cross-reference (i.e., a cross-node transaction is still required).
- **Inventory: Available-to-Promise.** Inventory can be distributed across shards, i.e., separate quantities of each item can be associated with each shard, effectively distributing access. But in the case of uneven access patterns, it becomes possible that users on one shard will exhaust their local inventory while users on other shards will still be able to acquire the item. *Note: This isn't just for e-commerce. If a game server has a limited number of high-value items, then all users of that game server must be on that same shard to ensure consistent data. But what happens if that game server becomes popular, and too many users log in? If that server can't be sharded, then that server will slow down dramatically, if not crash.*

Does your MySQL sharded application need ACID compliance (<https://www.clustrix.com/bettersql/acid-compliance-means-care/>)?

That means, does each transaction need to leverage up-to-date data, or is it okay for the data to have latency, i.e., be a bit “stale”? The former will require cross-node transactions to ensure the data is transactionally consistent. The latter can occur if data is being replicated across the nodes, because MySQL replication is asynchronous to the master transaction.

Once you confirm the business rules your applications with MySQL sharding must support, we can investigate the deployment options.

There are basically 3 cross-node transaction options for deploying MySQL sharded applications:

1. Don't support cross-node transactions
2. Cross-node replication to avoid cross-node transactions
3. Support cross-node transactions

Option 1: Don't support cross-node transactions when sharding MySQL

Pros: This is very simple from an application standpoint. The application treats each shard as an autonomous MySQL server. Thus, no application changes are necessary, and ACID compliance is maintained.

Cons: Since all data needed for a transaction must be co-located in the same shard, this requires careful data distribution across all the shards, to ensure all data directly related to the sharding key shares the same server. In addition, all inventory quantities need to be distributed (equally) across all the shards. Both of these require constant rebalancing, and still can result in hotspots and “feast or famine” depending on which shard the transaction is running. And finally, real-time lookups, analytics, etc., across shards won't be supported.

Option 2: Cross-node replication to avoid cross-node transactions

Another option is to create an array of cross-node replication processes between the shards, allowing each shard to have a local copy of data it needs to access. But since replication is read-only, this data access is limited to read-only access of that data, and has the potential of being stale due to replication lag.

Pros: Each shard can have a local copy of lookup data, allowing JOINS using the local RDBMS and not requiring any application changes. This allows more business functionality and avoids some of the MySQL sharding business challenges, despite not supporting cross-node transactions.

Cons: Since replication is asynchronous, the replicated data might not be consistent, i.e., it may be stale or wrong. For instance, if catalog descriptions are replicated, changes to the master won't be reflected in current/inflight transactions on other shards in the array. This can create two MySQL sharding business challenges: increased OPEX and database administration. Setting up cross-node replication across all the nodes in the MySQL sharded array is complex, requiring a lot more work by DevOps.

Option 3: Supporting cross-node transactions with MySQL sharding

If your MySQL sharded application needs to support cross-node transactions, then all the JOIN functionality—which automatically works in a single-node MySQL JOIN transaction—will have to be (re)created in the application itself. This is a very non-trivial task; relational database (RDBMS) theory and programming skill is a specialized niche, and not commonly experienced by the majority of application programmers. This means either new RDBMS-skilled programmers must be found and hired, or your internal team must be upskilled. This

represents significant risk. Whereas a widely deployed RDBMS like MySQL has had literally millions of QA hours spent refining its relational calculus and transactional ACID guarantees, when you “roll-your-own” you’re quite literally “re-inventing the wheel”... with all the risk (business, technical, financial, etc.) that implies.

Drilling down a bit deeper, a MySQL sharded application needing cross-node transactions will require application code to query one shard, then query a different shard, and then build its own relational logic between the two. If there are uniqueness or parent/child constraints needing to be enforced, those have to be (re)created in the application. Translation: if there are business rules restricting how many orders a customer can have in-flight, what kind of payment methods used depending on discounting, etc. Or on a game server, how many users can be in a certain area, how many critical/high-value items can spawn per location per unit of time, etc.

All these business rules potentially have to hand roll their JOINS across different shards, rather than doing a simple SQL query. Writing SQL (RDBMS query language) is tough enough to do accurately and performantly at scale. Adding in the requirement to ensure relational calculus can introduce all sorts of MySQL sharding business challenges including risk, expense, delay in deployment of (new) features, and exposing the application to data inconsistencies and/or corruption.

MySQL Sharding Business Challenge #2: A continually growing development team

Perhaps the biggest risk of MySQL sharding is that managing a complicated sharded array can gradually transform your DBA and IT teams into an “internal development team.” From the SVP of IT at a top-5 bank:

“We’re in the process of migrating from the west coast to our Austin data center, and the new team took one look at our 128+ node sharded systems and said “No, we’re definitely not going to support that”. And everyone on the project team signed-off, until the higher-ups got a look. “That system runs critical reports for upper management- you cannot turn it off”. And now everyone’s in a panic- they don’t have the resources to support that system in Austin, nor all the training needed.

“In short- with every new Feature that’s been requested to add to that system, we’ve had to update the sharding keys, modify the data distribution (using replication between nodes), and update the queries to take replication latency into account to avoid stale data. And that’s not even mentioning the ongoing shard maintenance, data/hotspot rebalancing, and ensuring we have full backups as close to the same point-in-time as possible.

“Our team is tasked to support Operations, NOT become a de facto development organization.”




Finding yourself with a continually growing dev team in your IT budget accounts for a *lot* of unintended, ongoing OPEX.


MySQL Sharding Business Challenges Summary

MySQL sharding is a well-known solution to provide scale, especially write scale,. However, hidden in that solution are not just challenges of technical complexity, but challenges to the flexibility of your business.

If your main data workload resides on a sharded MySQL array, some trade-offs will need to be made. Can your business do without cross-node transactions? Or make do with read-only lookups, which could be stale? Or do you really need cross-node transactions, and so you have to live with increased transaction latency... and significant application changes and ongoing maintenance (cf. “in-house IT dev team”).

*To learn more about how ClustrixDB avoids these MySQL sharding business challenges, read about our **shared-nothing architecture** (<https://www.clustrix.com/scale-out-architecture/>) and our **alternatives to sharding** (<https://www.clustrix.com/elastic-scale/>), and how **ClustrixDB scales-out both writes and reads** (<https://www.clustrix.com/resources/clustrixdb-rdbms-scales-writes-reads/>).*

 (<https://www.facebook.com/Clustrix>)  (<https://twitter.com/Clustrix>) 

(<https://plus.google.com/111948679378130082453>)  (<https://www.linkedin.com/company/clustrix>)

Product

Overview

(<https://www.clustrix.com/summary-of-our-db/>)

Cloud Database

(<https://www.clustrix.com/cloud-database/>)

Elastic Scale

(<https://www.clustrix.com/elastic-scale/>)

Scale-out Architecture

(<https://www.clustrix.com/scale-sql/>)

Solutions

Case Studies

Hit Labs

(<https://www.clustrix.com/resources/customer-success-story-hitlabs/>)

Match.com

(<https://www.clustrix.com/resources/customer-success-story-two-com/>)

Viverae

(<https://www.clustrix.com/resources/customer-success-story-viverae/>)

Recent News

MySQL-compatible

cloud database cost comparison

(<https://www.clustrix.com/bettersql/mysql-compatible-cloud-database-cost-comparison/>)

Ad Tech Carousel

(<https://www.clustrix.com/slideshow/ad-tech-carousel/>)

Gaming Carousel

(<https://www.clustrix.com/bettersql/gaming-carousel/>)

Support

(<https://support.clustrix.com/>)

Documentation

(<http://docs.clustrix.com>)

Blog (/bettersql/)

Resources

(<https://www.clustrix.com/resources/>)

Free Trial

(<https://www.clustrix.com/free-trial/>)

Privacy Policy

(<https://www.clustrix.com/privacy-policy/>)