Why Xpand

Is the write capacity of your single-instance MySQL database nearing its maximum? Is the CPU utilization of your present configuration at an all-time high? Has your system's performance slowed to a crawl such that users are complaining? Have you recently lost data due to unexpected hardware or network failures? Or is business booming and you anticipate encountering these issues in the near future?

Those warning signs can indicate that you are outgrowing your current configuration. There are numerous options to consider when evaluating your future system requirements. For example, expanding to a larger server may provide a temporary solution but eventually that path becomes no longer practicable nor economical.

The information below outlines some common upgrade alternatives, recaps potential obstacles of each, and provides justification as to why Xpand may be a viable alternative.

Option A - Use Replication to Expand Capacity?

Implementing either a Master/Slave or Master/Master replication topology to expand and distribute your system is one strategy many consider. In a Master /Slave configuration, the master typically performs writes, while the slaves service reads. As necessary, a slave can be promoted to a master. With Master /Master topologies, updates to multiple masters are synchronized and failover is built-in, as long as a single master can handle the workload. Both strategies allow reads and writes to be distributed amongst multiple servers to improve throughput or reduce logistical issues, and are certainly viable options for some implementations.

But one needs to be aware of the following caveats:

- Writes to masters don't scale beyond the capacity of the server they occupy. That means that once the server used for a master reaches its capacity, you'll need yet another solution.
- In a Master/Slave topology, reads from slaves can add read capacity, but slave lags can render stale results. Stale data is sometimes acceptable, but if your application relies on consistency, stale data can wreak havoc. For example, online purchases that need to access inventory levels need a consistent, accurate view of the data to correctly fulfill orders.
- The goal of implementing a Master/Master topology is typically to scale write capacity. Adding slaves to additionally scale read transactions can introduce multiple update lags - one for the master, then one for each slave. The resulting stale data may be problematic.
- Master/Master topologies introduce challenges to coordinate conflicting concurrent updates. For example, what should happen if multiple users update the same row of data on multiple iterations of a master with conflicting information?
- Bulk updates performed in databases that employ Master/Master replication can readily introduce inconsistencies between the masters due to synchronization lags.
- Each new master or slave added to the system adds overhead for synchronization and coordination.

Option B - Shard Your Data as an Alternative?

Another approach to address growth is to distribute data across several computers by use of sharding. With horizontal sharding, whole tables are divided into subsets (i.e. U.S. customers versus foreign customers) and spread across multiple servers. With vertical sharding, tables are split by columns (i.e. frequently accessed columns versus infrequently used columns) and placed on different servers. There are several sharding and database traffic managers available to help manage a sharded environment or you can build your own. But they all have the same potential pitfalls:

- Joining tables across multiple shards is often difficult, does not perform well, and frequently requires forwarding large amounts of data.
- Specialized coding is often required to ensure that applications know the location of the data.
- ACID compliance, transactionality, and referential integrity will need to be developed and maintained at the application level.
- Offloading that standard RDBMS functionality to an application layer can be expensive, risky, and can degrade performance.
- Operational tasks such as backups, schema changes, adding indexes to improve performance, and upgrading applications become unwieldy.
- Recovering all shards to a single point-in-time (backup recovery) is nearly impossible.

Option C - Is a Distributed Solution Needed?

There are numerous distributed solutions available (both SQL-based and NoSQL) that allow two or more servers to be combined or "clustered". Some of these distributed systems can be considered "shared nothing" as the servers do not share memory or disk storage. In a shared nothing environment, each server is self-sufficient and operates independently. Shared nothing architectures have been proven to be the best for scaling-out capacity and performance. Performance is enhanced because processing can be parallelized to multiple servers and there is no single potential bottleneck. One of the biggest challenges of distributed systems, however, is coordinating how the various processes, workloads, and data get distributed and balanced between multiple servers.

Some of the tradeoffs to be considered with a distributed solution include:

- NoSQL solutions often scale nicely but relax support of transactions, ACID compliance, and RDBMS referential integrity to do so.
- Some NoSQL products now support SQL language semantics, but that does not mean they support typical RDBMS guarantees such as ACID compliance, real-time transaction processing, and referential integrity.
- Some distributed solutions are add-ons to a single instance database that emulate the functionality of a distributed system. Such solutions frequently do not scale for writes as robustly as they do for reads.
- Others may require expensive refactoring and customized changes to your application.

Option D - Is Xpand the Answer?

Xpand eliminates the need to consider any of the options noted above! Our clustered solution offers easy, flexible scaling for growth. A feature of Xpand to Flex Up and Flex Down a cluster allows your database to adjust simply by adding or removing node(s).

Xpand's shared nothing architecture accommodates true distributed processing without sharding. Distributed query execution promotes parallel processing by sending queries to the data versus bringing the data to each query. All such distribution is completely transparent to the system's users and accessible via a simple MySQL compatible SQL interface. No application customization is necessary to access this distributed data via distributed queries. The documentation relating to Distributed Database Architecture explains the numerous elements of Xpand that work together to make this possible.

Additionally, Xpand is fault tolerant. The database includes an inherent replication strategy that maintains redundantly updated copies of all data. In the event of an unexpected node failure, Xpand will automatically recover and continue operations with the remaining nodes. A Xpand cluster can lose a node without losing any data. To learn more, read about Xpand's Consistency, Fault Tolerance, and Availability.

Xpand provides these capabilities while maintaining conventional RDBMS properties such as ensuring real-time referential integrity, accommodating multi-table joins and ad hoc queries, and guaranteeing ACID Compliance. Xpand was designed to be a scalable drop-in replacement for MySQL.

Finally, Xpand provides fast, complete, consistent backup and restore features along with online schema changes that are made without service interruption.

Comparison Chart

Refer to the chart below to see how Xpand compares to the alternatives discussed.

Legend
✓ = Supported
✗ = Not supported

	Xpand	MySQL	Sharded RDBMS	NoSQL (Distributed)
Easily Scalable	✓	✗	✓	✓
Shared-Nothing Architecture	✓	✗	✗	✓
Distributed Processing	✓	✗	✗	✓
Applications Easily Ported from MySQL	✓	✓	✗	✗
Built-In Fault Tolerance	✓	✗	✗	✗
Built-In High Availability	✓	✗	✗	✗
Full-Featured RDBMS				
SQL	✓	✓	✓	✓
Referential Integrity	✓	✓	✗	✗
Multi-table Joins	✓	✓	✗	✗
Ad Hoc Queries	✓	✓	✗	✗
ACID Compliance	✓	✓	✗	✗

To obtain a free trial of Xpand, please contact Xpand Sales.