Evaluating ClustrixDB Compatibility and Performance

This section provides guidelines to aid customers in evaluating ClustrixDB for a given environment or application. These suggestions may help you evaluate how well ClustrixDB performs while simulating your production workload.

Evaluating Compatibility

If you are currently using MySQL, you can use the following methods to validate your application's compatibility with ClustrixDB:

- To validate **write** statements, set up ClustrixDB as a slave using the SBR mode of replication to ensure the ClustrixDB slave can handle your application's write queries.
    - SBR provides the simplest option to validate compatibility, but for a production deployment, RBR is recommended.
- To validate **read** statements, SQL queries may be captured either using tcpdump utility or by enabling full query logging.
    - Once you have captured tcpdump output, Clustrix Support can help convert it into valid SQL, which you can then replay using MySQL client.
- To understand how ClustrixDB differs from MySQL review the document General Differences from MySQL.

Prerequisites for a Successful Evaluation

1. Please ensure that your cluster is installed according to the Best Practices for ClustrixDB Platform Configuration, and that sufficient resources have been allocated for your tests. Note that the minimum supported hardware configuration is adequate for testing compatibility, but is not suitable for load or performance testing.
2. Check ClustrixDB and system logs for any recurring errors.
    a. These could include environmental issues, such as transient network errors or storage subsystem issues, which could introduce inconsistencies in performance or cause test failures.
3. It is best to perform one aspect of performance testing at a time to obtain consistent, comparable numbers. For example, unless you are specifically measuring mixed workload behavior, perform analytic query tests separately from OLTP tests.

Testing OLTP Performance

This section describes considerations when testing OLTP performance, where we are focused on finding the maximum throughput of the system while maintaining a reasonable response time (query latency). Given an appropriate test harness and set of queries, you can:

- determine whether the TPS/latency figures of a given configuration are suitable for your application.
- directly compare the performance of ClustrixDB and an alternative database using the same workload at varying concurrency.

Test Harnesses

Since ClustrixDB is optimized for high concurrency OLTP workloads, a test to exhibit this capability must simulate many users accessing the database at once. A single-threaded test (for example feeding mysql client a list of queries) will fail to utilize the distributed resources of a ClustrixDB cluster and obtain results no better than a single-instance database. A suitable performance test utilizing tens or hundreds of threads, however, will allow ClustrixDB to leverage its distributed architecture.

It is important to have some kind of test harness to run a distributed load against the cluster. A suitable test harness for high-throughput transaction testing should have the following capabilities:

- ability to run a number of queries against the cluster, measuring throughput (TPS) and latency (average query time, in ms).
- multi-threaded; ability to increase concurrency of the workload.
- light-weight and/or distributed across multiple clients.
- ability to maintain a balance of sessions across all nodes of the cluster

As with any performance exercise, it is important to identify and eliminate any bottlenecks. If your test harness is itself the bottleneck, you will not be able to differentiate the performance characteristics of the database under test. Make sure that your test tool is efficient enough that it can drive high load without itself requiring lots of CPU or memory; this is a particularly important consideration if you are building a test with a robust

application server framework, where you may find you need to allocate a large number of servers to adequately drive the database backend.

We have worked with customers who have successfully utilized existing load test frameworks such as YCSB and Apache JMeter, both of which are designed to scale load. We have also assisted users in utilizing sysbench to test performance and scale.

Distributing Client Load

It is important that your test includes sufficient concurrency (multiple threads) to engage all cluster resources and adequately model production load. Further, clients should distribute their thread connections evenly across all the nodes in a cluster. Some test harnesses (including sysbench) allow specification of a pool of hostnames/IPs to facilitate this. Otherwise, a front-end load balancer can be used, as described in Load Balancing ClustrixDB with HAProxy. You can confirm that load is evenly distributed during your test run by checking the output of The CLX Command-Line Administration Tool stat command, which shows TPS for each node, or with a query such as:

```
sql> SELECT nodeid, count(1) FROM system.sessions GROUP BY nodeid ORDER BY nodeid;
```

To optimally balance the load across the cluster, consider using a thread count that is an even multiple of the number of nodes. However, for sufficiently high thread counts (greater than the total number of cores in the cluster), exactly even distribution of threads will matter much less.

Test Data Set

For an existing application, a dump and restore of the current production data set provides the simplest data set for testing. For very large data sets, it may be desirable to utilize a subset of production data, in which case:

- mysqldump --where flag can be used to extract a subset of production data, including the neat trick --where "1 LIMIT 1000"
- Modify queries as necessary to operate on this subset of data (i.e. adjust a date range).

Please see Loading Data Onto ClustrixDB for guidance on optimizing data import performance.

Query Set Used for Testing

The ClustrixGui Administration UI will indicate if your test workload is dominated by a single query. You can also look for such a problem by running SHOW PROCESSLIST or selecting from system.sessions while running your test, to identify any long-running queries. If you discover queries that are dominating your workload, you can optimize them as described in Optimizing Query Performance.

Testing Analytic Query Performance

For OLAP testing, customers are typically concerned with query response time for complex queries, rather than aggregate throughput. In this case, a simple framework that executes queries one at a time is reasonable, but consider the following guidelines:

Execute Queries at Least Twice

Running the query for a second time eliminates two confounding factors, caching and query compilation. Depending on your dataset size and workload, most production queries may execute from cache, in which case cold cache performance is less relevant. ClustrixDB caches the compiled program of each query, so subsequent executions need not be recompiled. For particularly complex queries (i.e. with many-way joins), compilation time may comprise a significant portion of total execution time. Running a second time will give you a good idea of how the query will execute in production, where the compiled query plan will typically be cached.

Use Meaningful Queries

This goes hand in hand with the comments above regarding the use of a data set with a meaningful distribution of values. Make sure that your queries match your data. For example if you use a dump that is several months old, but your queries are taken from a current workload; make sure the date ranges in the queries match the dates in the data or the queries can return no rows.

Ensure Indexes are Available

The indexes necessary for the efficient execution of your queries are in most cases the same as needed for MySQL or other databases. However, given the distributed nature of ClustrixDB, the lack of an index can sometimes impose a more severe penalty than on a single-instance database.

This is due to broadcasting among nodes, which is avoided when proper indexes (i.e. on the columns of a JOIN clause) are available. Customers sometimes use queries from their MySQL slow query log to test against ClustrixDB. In some cases, we found that these queries were slow on MySQL due to lack of a proper index, and without the index, ClustrixDB was slower than MySQL. With the proper index, MySQL's performance was improved, but ClustrixDB with the index was even faster.

For more information on identifying and correcting queries that fail to use an index, see Optimizing Query Performance.

Recognizing Platform Resource Constraints

When testing the performance of your ClustrixDB cluster, consider the limitations imposed by your platform: CPU, memory, storage I/O, etc. The good news here is that, as a scale-out solution, the limitations of a single server can be overcome by growing the cluster to more nodes. However, to ensure growing your cluster will improve the performance of your workload, it is important to identify the limiting resource constraint.

Recognizing Platform Limits provides detailed information on identifying resource constraints, but to summarize:

- If average CPU load for your cluster is approaching 80-100%, your workload is CPU-bound and will benefit from additional nodes.
- Else, if buffer manager miss rate is significant (2% or more), and disk latency is high, your workload is likely disk-bound.
    - If your working set can be kept in cache through the additional memory made available by expanding the cluster to more nodes, adding these nodes can improve performance significantly.
    - If you have high disk latency, confirm you are using local SSDs.
    - Note that bad plans caused by missing indexes can also increase BM miss rates. See Query Optimizer for additional information.
    - While uncommon, check that the network between nodes is not becoming a bottleneck (use system.internode_latency). Typically this would happen due to misconfiguration or the provisioning of cluster nodes from different subnets.

Testing Scale-Out

To validate the scaling capabilities of ClustrixDB, you may wish to repeat your tests, whether OLTP, analytics, or both, with different cluster sizes. When doing such scale testing, particularly with a smaller data set, it is important to have your data evenly distributed for each iteration of cluster size. Our recommended approach is as follows:

- Start with a cluster that is larger than you need, and try to achieve desired target TPS/Latency.
- For very large tables (> 100 Gigs) that will keep growing, import the data into 8 Gig slices with one slice per core using clustrix_import wit h the -m option where -m is the number of slices. Without the -m option clustrix_import ensures each table/index has at least once slice per node.

```
shell> clustrix_import -u<user> -p<passwd> -i <dump_filename> -D <databasename> -m 48
```

- Create smaller tables with a slice count that is a common multiple of the cluster sizes you plan to test (e.g. 18 if you plan to test 3, 6, and 9 nodes).
- Once your tests are running within acceptable limits, scale cluster down to the point where it can sustain the load at < 80% average CPU.
- If for some reason you need to grow the cluster again, you may need to re-slice your tables (ALTER TABLE <table_name> SLICES=<number of nodes>) after growing the cluster.
- Check that the Rebalancer is finished rebalancing and that system.layer_merges are complete, after reducing or expanding the cluster.

For guidelines on tuning the Rebalancer, see Managing the Rebalancer.