TOSHIBA
Leading Innovation

GridDB Reliability and Robustness

April 21, 2017
Revision 1.0.6

GridDB™

Table of Contents

List of Tables and Figures

Executive Summary

Fixstars explains the reliability features of Toshiba's GridDB NoSQL database software and how in practice they work.

Introduction

Although big data has grown and databases have begun to scale-out by creating distributed networks of inherently unreliable servers, the market still demands error-free, 100% availability. Database software has now become the key to providing scalability and reliability.

Typically, database reliability is defined as two factors, availability -- or how infrequently the database is offline -- and consistency -- which means returning the correct result. GridDB can be both highly available and consistent depending on its configuration.

This white paper examines the design of the GridDB software as described in the GridDB Technical Design Document[1] and the mechanisms in place that allow GridDB to both scale-out and remain reliable. The three main facets of GridDB's reliability are:

- Hybrid Cluster Management Architecture
- Partition Replication with the Advanced Data Distribution Algorithm.
- Client Robustness

These mechanisms are also demonstrated, providing an example on how developers and administrators should expect their GridDB cluster to behave in the event of a failure.

Reliability Features

Like many other distributed databases, GridDB has several features designed to ensure it can maintain high availability, such as partition replication and client robustness. Unlike many distributed databases, GridDB has a unique hybrid cluster management architecture that provides the excellent performance of a master/slave configuration, with nearly the same fault tolerance of a peer-to-peer architecture.

[1] http://www.griddb.org/griddb_nosql/manual/GridDBTechnicalDesignDocument.pdf

Hybrid Cluster Management Architecture

There are generally two types of clusters: master/slave and peer-to-peer.
A master/slave cluster has a single node or single set of nodes that act as a master, which orchestrates the operation of the remaining slave nodes. This architecture has little overhead but the master node presents a single point of failure as well as a choke point that can limit the scalability of the database. This makes master/slave clusters good for performance but are not as reliable. MongoDB, for example, uses fixed masters called primary nodes.

In a peer-to-peer cluster, every node is identical and has the same responsibilities. In the case of node failure, the remaining nodes can easily take over for a failed node, though overhead to maintain consistency is quite high. Peer-to-peer clusters are good for reliability but the architecture can suffer from performance problems, especially without fine tuning. Cassandra uses a peer-to-peer cluster topography.

GridDB has a hybrid cluster architecture. To quickly summarize, all nodes in a GridDB cluster are identical and one node is elected master. If the master fails, any of the remaining nodes can take over.

On start up, the master is initially elected using a bully algorithm where all nodes send messages with their ID and the node with the highest ID wins. The master then receives heartbeat messages from the follower nodes.

Each node in a GridDB cluster has two roles, it can be a master or follower and a partition owner, backup, or catch-up. The master is the owner of the partition table and the partition table is replicated to all the follower nodes. In the case of a master failure, one of the followers is promoted to being the master.

Partitions are logical blocks of data that store containers. The owner of a partition can both read and write the partition while the backup only permits direct read operations. A catch-up node stores a replica of the partition but clients will not refer to it until it's promoted to a backup node.

When a follower does not receive the master's heartbeat, the follower can then call an election, and the same bully algorithm that was used to initially determine the master is utilized again to determine the replacement master. Since all followers have a replica of the partition table, the cluster resumes normal operation within seconds.

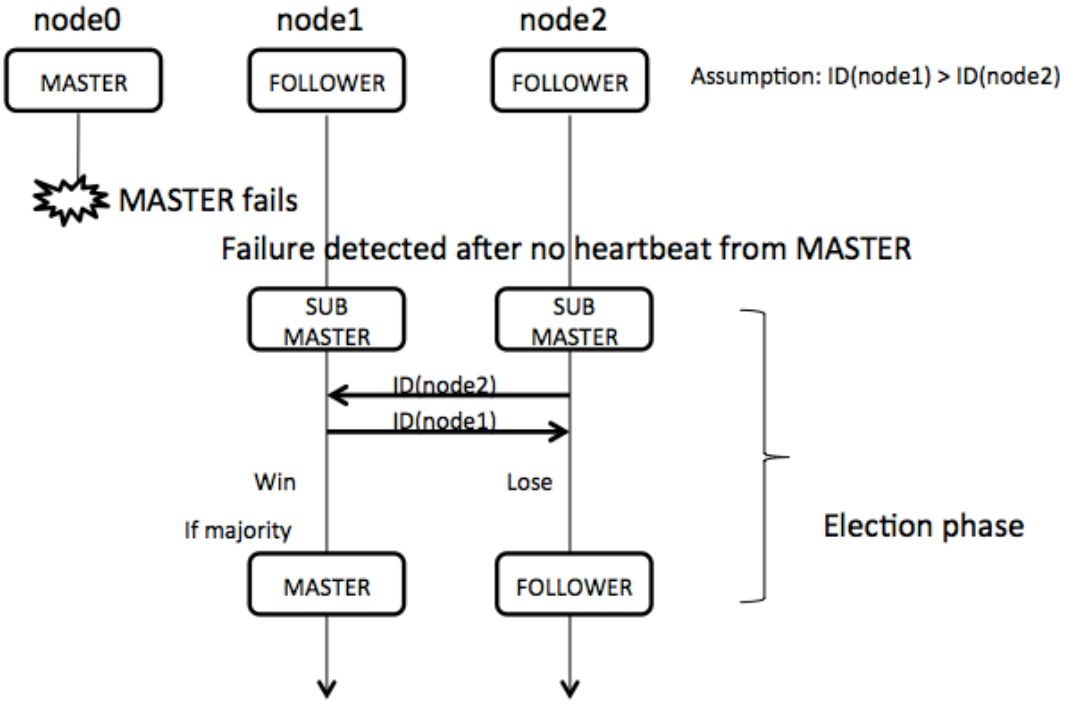

Figure 1: Master Failover Sequence

Partition Replication

Replication is the primary factor in how scale-out databases can provide high reliability. Essentially, replication stores each record on multiple nodes within the cluster.

In GridDB, the number of replicas can be configured as part of the gs_cluster.json file. If the number of replicas is set to one, there will be a loss of availability if any node in the cluster fails, but if the number of replicas is set to three, any two nodes may fail without a loss of data or availability. Of course, setting replication to three means that the owner and two backups will store the data, consuming three times the amount of resources required if replication was set to one.

The Autonomous Data Distribution Algorithm (ADDA) controls how and where replicas are written while performing background synchronization, allowing GridDB to scale without interruption.

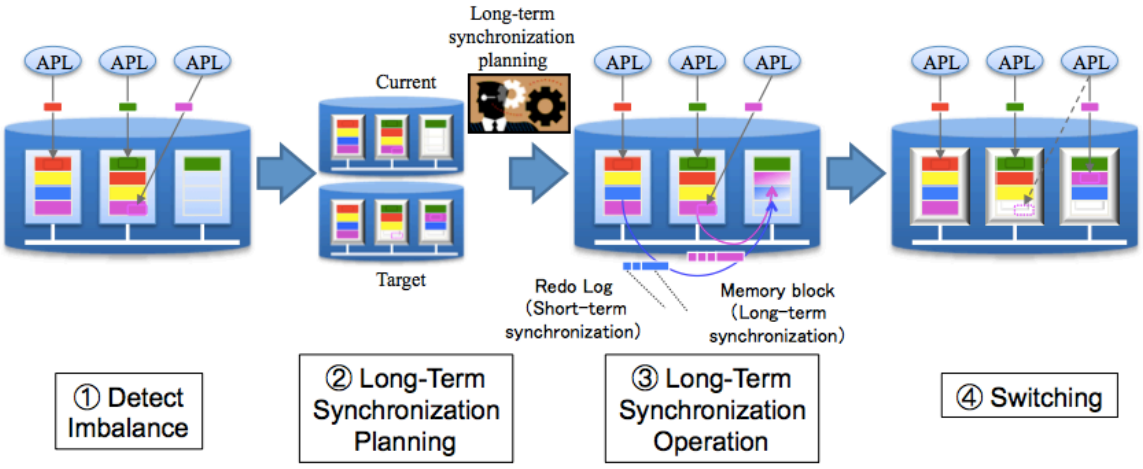

Figure 2: Autonomous Data Distribution Algorithm (ADDA)

Figure 2 demonstrates ADDA's Long-Term synchronization. In order to reduce the load, long term synchronization is executed on a fixed interval (usually 1 minute) in the background.

Short-Term synchronization is activated when all logs after the current edition are available. The amount of data sharply increases due to short-term synchronization, but processing is interrupted after 30 seconds (default).

Writing Replicas

There are two methods in which GridDB may write replicated records, asynchronously or semi-synchronously:

Asynchronous updates will first update the owner of the record and then return to the client before updating the backup nodes. This method provides good performance but can lead to consistency issues if the owner of the record fails before the backup nodes are able to process the update.

Semi-synchronous updates are processed first by the owner and then all the backup nodes before returning to the client. The initial operation will take longer than if it was done asynchronously, but the records will always be consistent between the owner and backup results.

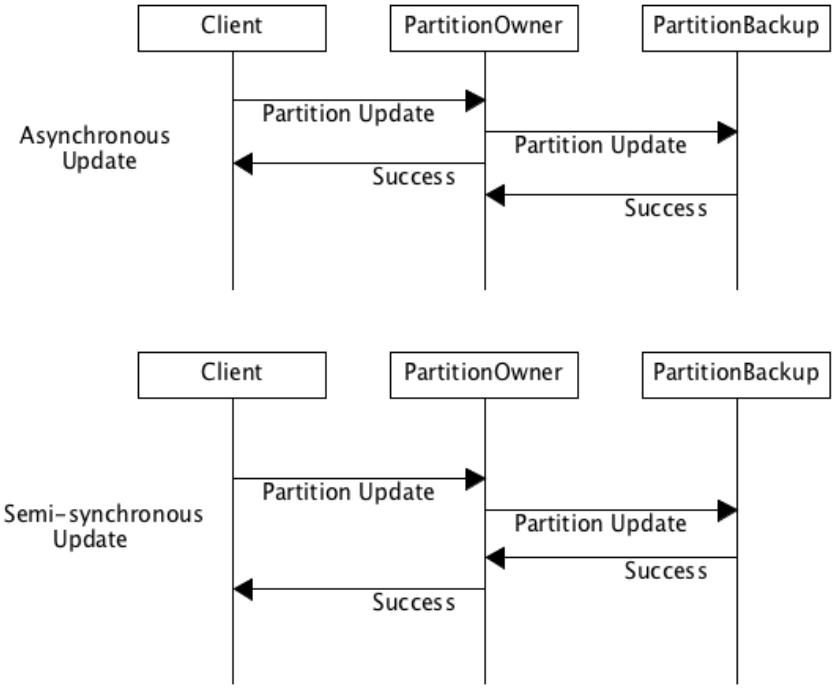

Figure 3: Difference Between Asynchronous and Semi-synchronous Updates

Failover

Failover of a partition owner occurs when the master has not received a heartbeat from the owner of a partition in the specified time. The master then chooses a new owner from the available backup nodes and a new back up node from the available catch up nodes and distributes the new partition table to all of the followers.

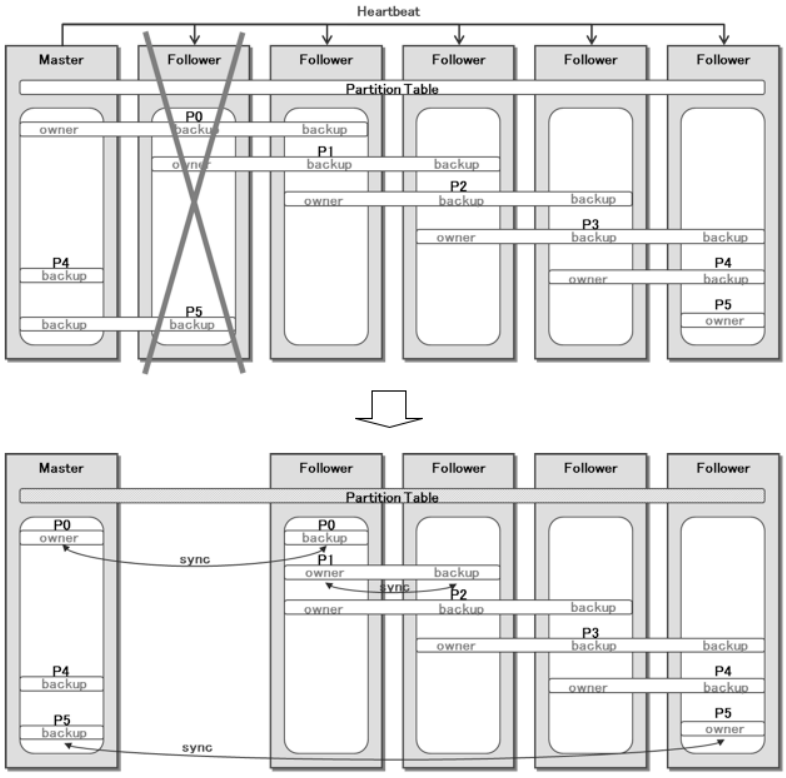

Figure 4: Synchronization after Node Failure

The new owner and new backup will then confirm their update log is identical, and if it is not, they will synchronize. Since the deltas between update logs will be fairly small, this short-term synchronization should be fairly fast and complete within seconds.

Replacing a Node

When a new node connects to the cluster, it will be assigned the role of a catch-up node, which is essentially a backup node's backup. After the initial synchronization is complete, it can be promoted to a backup role (and then to an owner role) as required.

Catch-up nodes will be promoted to backup nodes when a node fails, there are an insufficient number of backup nodes for the replica setting, or an inconsistency exists between the partition owner and back up nodes.

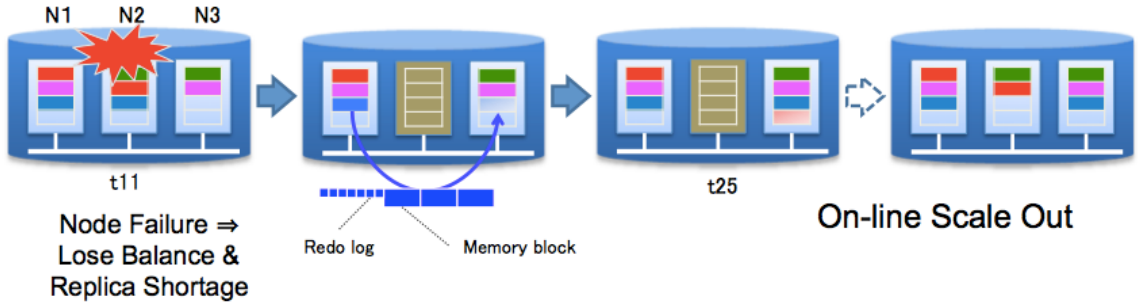

Figure 5: ADDA Maintaining Replication After a Failure and Replacement.

In Figure 5, ADDA starts synchronizing partitions between N1 and N3 after N2 fails. When N2 is replaced with a new node (N2'), it is able to rebalance the partitions between nodes so N2' stores a relative number of records.

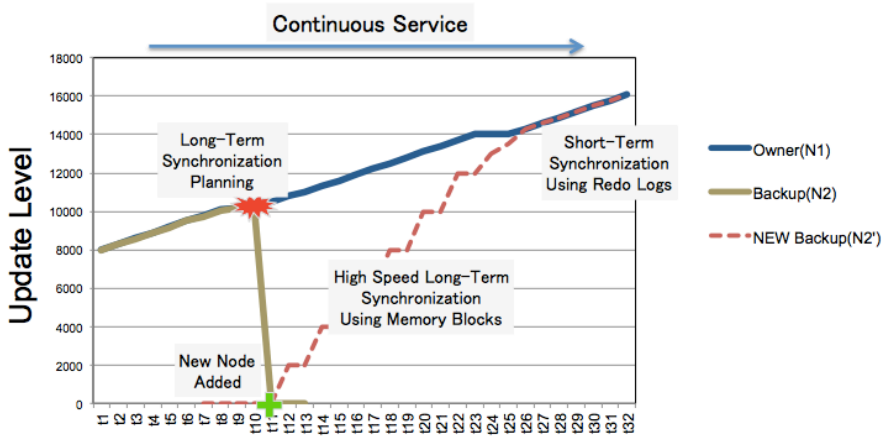

Figure 6: ADDA Synchronizing Records

Figure 6 demonstrates the two-phase synchronization ADDA performs after a node failure at t11 while data in consistently added or updated. Initially, large memory blocks are synchronized which allows for the replacement node to catch up to the owner quickly and then at t25 Redo Logs are transferred and applied. This two-phase approach allows the Owner to continue to process request updates while the New Backup is synchronizing.

Client Robustness

The previous sections of this white paper have focused on reliability within the cluster of nodes, but to have a completely reliable system, the client itself needs to be robust enough to tolerate faults to its connections to the cluster. It also needs to tolerate individual nodes failing as they communicate directly with partition owners.

Initially, the client will receive a message from the master node that specifies the master and then the client will also cache the partition table, which it retains until there is an error.

This way, there are minimal requests sent to the master which will prevent the master from becoming a choke point. Furthermore, a node is specified with reference to a partition table of a master node, and a statement is performed.

The client library has a client failover mechanism that enables the client to withstand failure if there is a node or network failure. If a connection is closed or the partition checksums do not match between the client's cache and the node, the cache is invalidated and the statement is rerun until it succeeds. The client library only returns an error to the application if the timeout is exceeded.

Real-World Testing

To perform real world testing, Fixstars constructed an 8-node cluster where 3 nodes would run GridDB and the other 5 nodes would run YCSB[2] to perform a basic workload. GridDB Standard Edition version 2.9.0 was used on the GridDB nodes.

Follower Failure

After GridDB was started an insert-only workload was run on five nodes once the server processes settled into a stable state and throughput was monitored for the entire duration of the test.

After a set amount of time, the database server process a random GridDB follower was killed while throughput continued to be monitored.

The following table shows normalized throughput over the course of the test.

[2] https://github.com/brianfrankcooper/YCSB

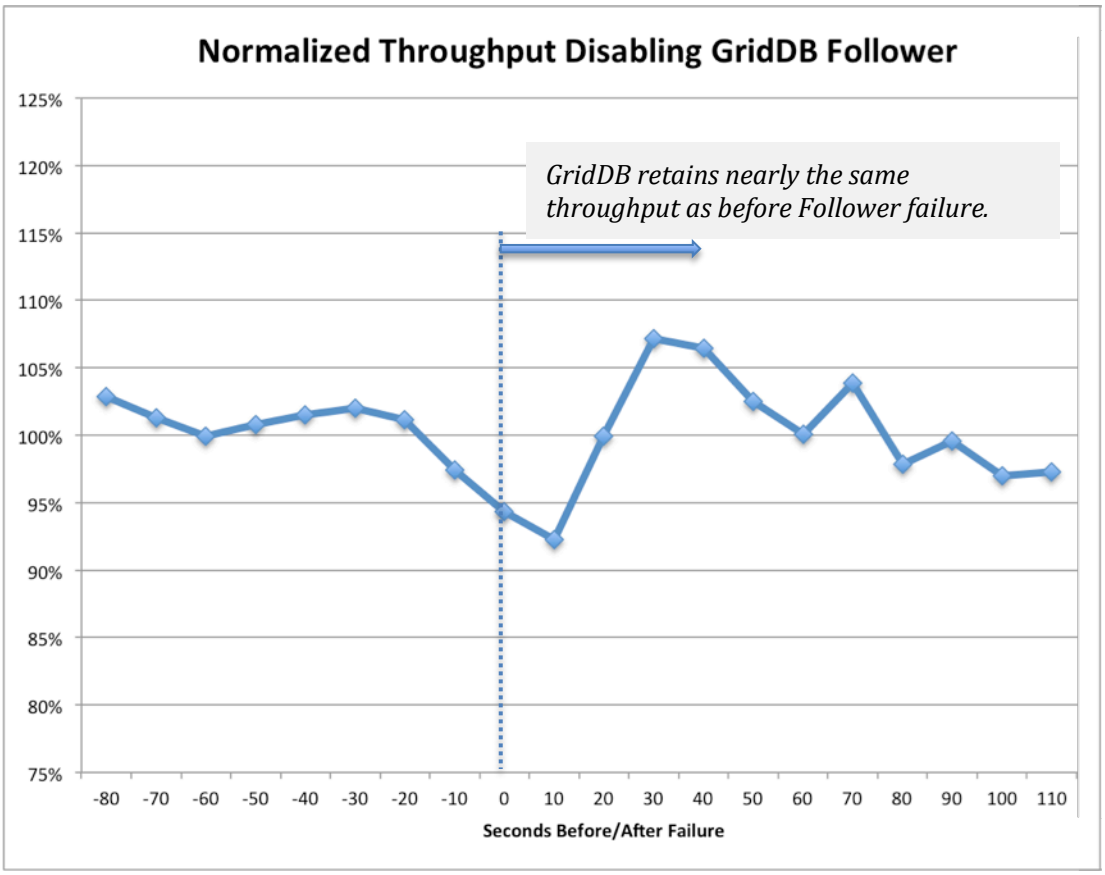

Figure 7: Normalized Throughput Disabling GridDB Follower

When a GridDB follower fails, throughput drops slightly before quickly recovering and is then able to maintain the same throughput as prior to the failure.

Master Failure

After GridDB was started an insert-only workload was run on five nodes once the server processes settled into a stable state and throughput was monitored for the entire duration of the test.

After a set amount of time, the database server process on the GridDB master was killed while throughput continued to be monitored.

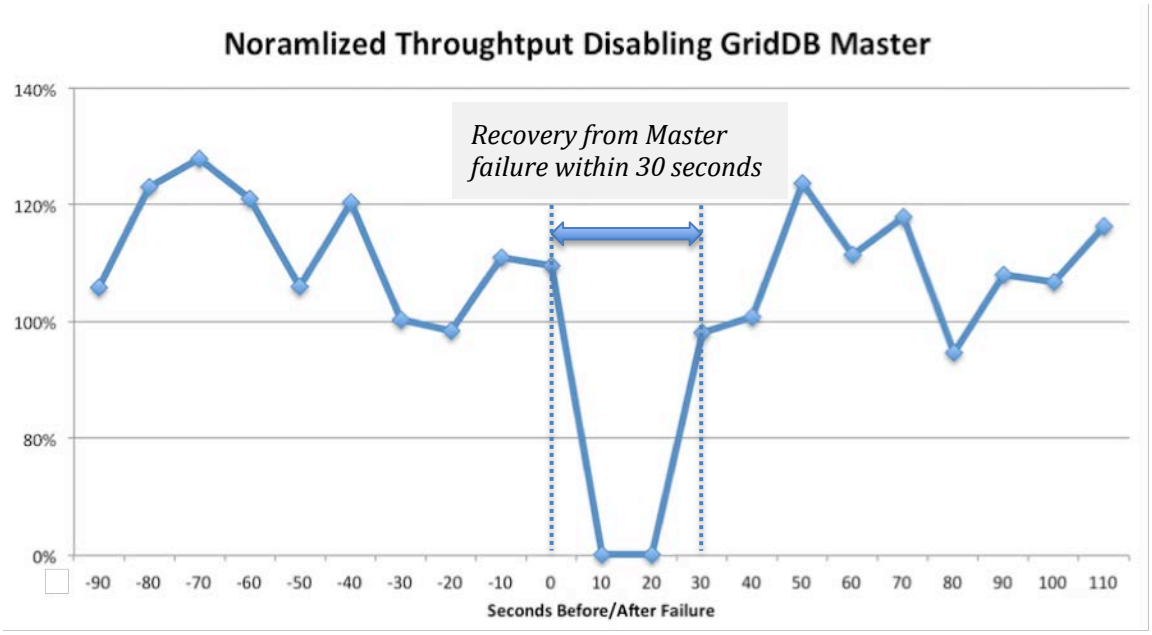

Figure 8: Normalized Throughput Disabling GridDB Master

In most master/slave applications, the master failing would lead to significant downtime, but GridDB recovers from the master failing within 30 seconds and resumes processing requests at a similar throughput pre-failure.

Failure Latency

While running the Master and Follower failover tests, latency data was also collected. The data demonstrates absolute and relative response times during a node failure

	Average	95% Percentile	99% Percentile	Max
No Failure	487us	998us	1,710us	92,699us
Follower Failure	495us	961us	1,837us	1,763,890us
Master Failure	578us	1,052us	2,495us	21,643,263us

Table 1: Request latency during node failures.

The total runtime for the YCSB workload is approximately 5 minutes with the failure occurring approximately 100 seconds after YCSB begins. Examining log files show that the GridDB cluster is recovering for approximately 30 seconds or 10% of the total duration. This would mean that both the 95% and 99% Percentile Latencies would include the duration of the recovery period after the failure is triggered.

Adding a New Node

In this test, a database cluster consisting of two servers configured with a replication level of two were started, with 2GB of data loaded with YCSB; a third node is then added to the cluster. The statistics of the cluster were then monitored while the data records from the initial two nodes were transferred to the new node.

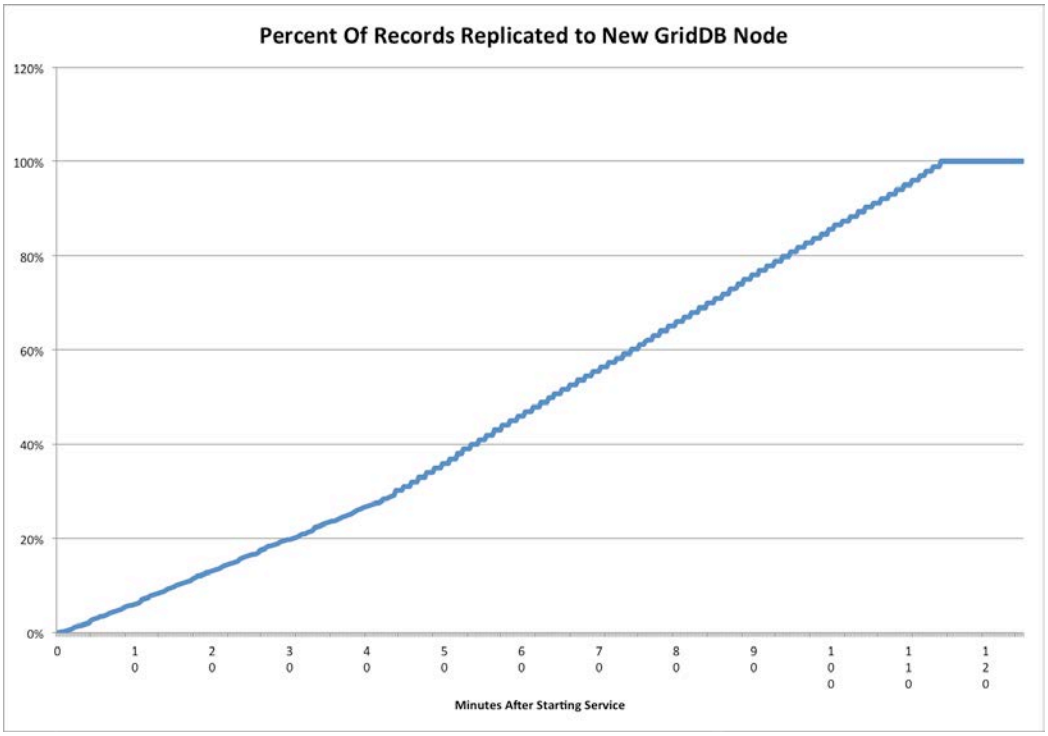

Figure 9: Percent of Partitions Replicated to a new GridDB Node

Figure 9 demonstrates ADDA performing long-term synchronization after a third node is added to a two-node cluster with replication set to two.

Long Term Synchronization is executed on one-minute intervals causing the amount of synchronized data to increase step by step. Since the node is added to a cluster there is more than 90 segments corresponding to $1/3 \times 2 = 2/3$ of the total number of partitions (default is 128).

Once synchronization of the required partitions is completed, the three-node cluster is in a stable state.

Conclusion

GridDB is a scale-out database that is both fast and extremely reliable with Hybrid Cluster Management Architecture, Partition Replication, and Client Robustness features that have demonstrated efficacy with the error injection testing performed by Fixstars. Meanwhile, GridDB's Autonomous Data Distribution Algorithm ensures that after a failure or upgrade, the data stored in GridDB is stored in a balanced manner.