Building IoT Applications with GridDB

September 28, 2017
Revision 1.0

Table Of Contents

List of Figures

1 Executive Summary

This white paper showcases the requirements of IoT (Internet of Things) databases by examining the architectures of IoT solutions and how the data is generated, transferred, and used after it has been collected. Finally, it demonstrates how GridDB is specifically tailored for these different scenarios and can ease both initial development and on going maintenance.

2 Introduction

It is estimated that by 2020, there will be 50 billion devices connected to the internet [1]. All these devices will be connected to the cloud, each other, and to different services. This will create a new dynamic and global infrastructure known as the "Internet of Things." This infrastructure will completely transform how individuals and organizations connect to each other.

Comprehensive data management is key for many IoT applications as many decisions and services are based on the various ways to combine both real-time and historical, stored data. One major component to consider in designing an IoT data management framework is choosing its database. IoT databases have a much different set of requirements when compared to the enterprise systems of the past.

Toshiba's NoSQL database GridDB was originally designed for IoT workloads and provides the performance, flexibility, reliability, and support needed for such applications. GridDB is a scale-out, partitioned database that features include in-memory storage and processing for high performance and scalability. It has a flexible key-container data model that can be easily adapted for use for a variety of different data types. Its use of partitioning and a hybrid cluster management architecture provides high availability with reliability. It also provides wide support of popular programming languages and third-party software packages to make analyzing data and building applications easier.

[1] https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/iot-platform-reference-architecture-paper.pdf

3 IoT Infrastructure

To understand IoT data, first we should understand the physical components in an IoT solution and how they're used.

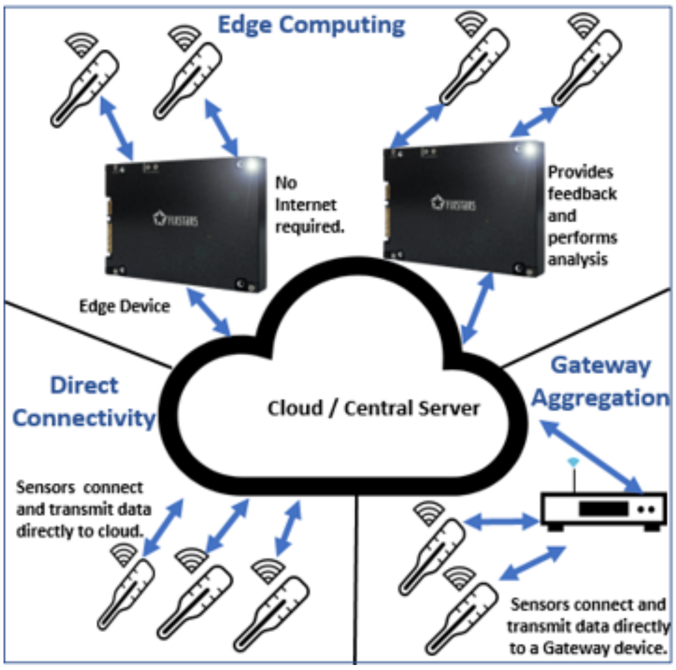

Edge Computing
No Internet required.
Provides feedback and performs analysis
Edge Device
Direct Connectivity
Cloud / Central Server
Gateway Aggregation
Sensors connect and transmit data directly to cloud.
Sensors connect and transmit data directly to a Gateway device.

Figure 1: Visualization of IoT Models

3.1 Sensors and Actuators

Sensors can be defined broadly as devices that provide inputs about its current state while actuators are devices that are used to make changes in the environment. IoT devices can range from small inexpensive microcontrollers to expensive industrial machinery. There are thousands if not millions of sensors in a typical IoT platform each generating data on a regular interval.

3.2 Communication

The communication component of an IoT application transfers all the data between the sensors, gateways, and data center or cloud. The components that are connected to each other and how they are connected can differ. One model involves connecting sensor devices directly to each other for communication. The cloud-to-device model has the IoT device connected directly to a cloud service or private data center to allow for services like remote access. The gateway-to-device model connects sensor devices to an intermediary device known as a gateway. The gateway can add extra interoperability between cloud services and IoT devices.

3.3 Analytics

The intelligence component of an IoT application is the portion that stores, analyzes, and process vast amounts of data. It consists of various technologies and frameworks such as databases and data processing frameworks and utilizes cloud computing. The operations of data processing usually consist of aggregation, analysis, and storage.

IoT applications can be deployed in a variety of different ways for various domains. The layouts can focus on certain layers or endpoints of the application. Different layouts and architectures have different requirements. The various IoT application structures also differ in their size, scope, and the number of domains that they encompass. The primary structures of an IoT system are Edge Computing, Gateway Aggregation, and Direct Connectivity.

3.4 Edge Computing

The edge is the location where all event data and automated action takes place. In edge computing, there are three device types: the edge gateway, the edge device, and the actual edge sensor[2]. Edge devices can be thought of as general-purpose devices with full operating systems and processors. These devices do not require internet connectivity and can perform analysis and feedback on their own. Their usefulness come into play when the data volume is so large that a central server cannot handle all the data at once. They are also useful to make data processing as close to real-time as possible. The edge gateway has a full operating system with higher computing resources than the edge device. The gateway acts as the intermediary between the central server and the edge device.

An Edge Computing IoT device will process or filter its data before propagating the portion to be stored to the GridDB database. That data can be propagated either using GridDB's native APIs or via some other messaging framework such as MQTT.

3.5 Gateway Aggregation

An intermediary gateway collects information from a set of local sensors and then aggregates their data before sending it to a central server. Gateway devices are useful for bridging different network types and are typically used in tightly coupled systems, for example all the sensors within a building would communicate with the building's gateway which would send the data to the centralized server that receives data from multiple gateways.

A gateway device can either directly connect to a GridDB cluster using native APIs or it can send data via a secondary messaging protocol to the datacenter or cloud where the data would be ingested.

[2] https://www.ibm.com/blogs/internet-of-things/edge-iot-analytics/

3.6 Direct Connectivity

IoT sensors and devices can directly send their data to the cloud or centralized servers. The centralized infrastructure can exchange data and control message traffic.[3] This style of communication is useful for loosely coupled systems like a network of smart meters. The devices can either directly connect to GridDB via its API or messages can be sent via an application gateway in the centralized infrastructure.

[3] http://www.thewhir.com/web-hosting-news/the-four-internet-of-things-connectivity-models-explained

4 IoT Data

Raw IoT data is unique in that it is typically machine-to-machine data that is generated continuously while being never directly used by a human. Its unique characteristics prioritize transaction count over raw data size while read performance and high-speed search are important to transform data into something useful via real time processing, batch jobs, or ad-hoc queries.

4.1 Record Size and Count

Typically, the records created by an IoT device are quite small, generally being only a few bytes in length, but are generated frequently and by a large number of devices. A few bytes per record quickly become a terabyte -- or even petabyte -- level problem requiring both a huge amount of storage and incredibly fast per transaction times.

To solve this issue, GridDB uses both in-memory and persistent storage. A singular transaction can be completed in memory very quickly and then grouped together over a defined period to be written to disk as one batch.

It aims to keep most or all its data in-memory, using checkpoint intervals to flush its internal memory structure back to disk. Affinity functions make effective use and operation of limited memory areas in a database.

GridDB provides the high performance and scalability required by IoT applications through horizontal scalability. Using a memory first architecture helps to maximize performance when ingesting and processing large amounts of data. GridDB's approach to scale-out support for adding additional nodes online give IoT applications the needed horizontal scalability. These features are further demonstrated in a YSCB benchmark test against Apache Cassandra.

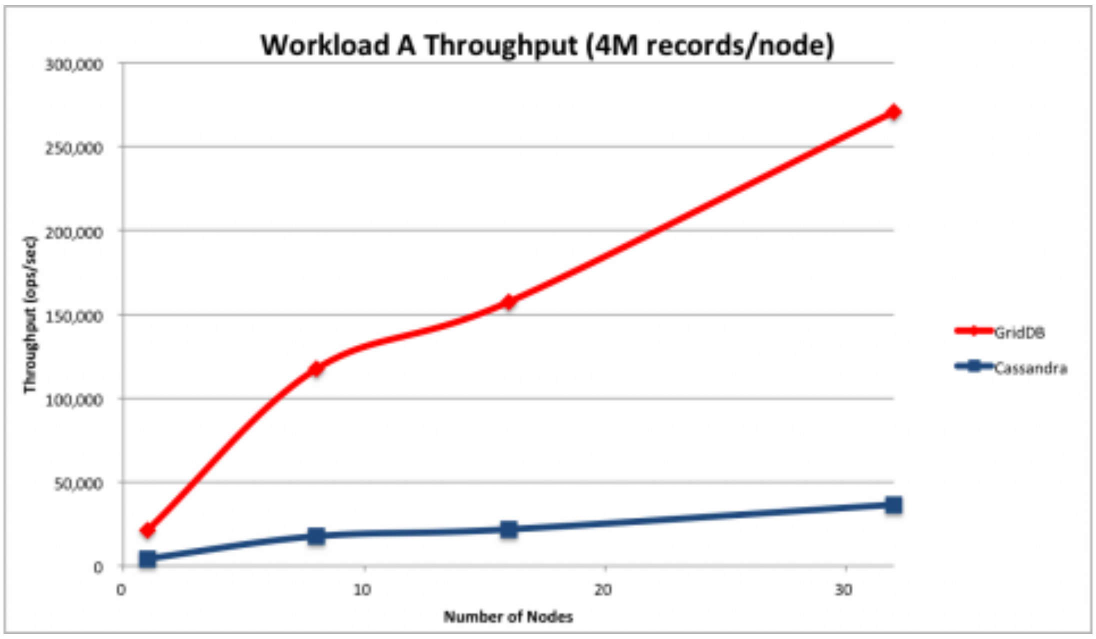


Figure 2: Difference in Scalability between GridDB and Cassandra

4.2 Database Growth

When an IoT platform first goes online, there is typically just a subset of sensors and data that it will have at full maturity. As the number of sensors and amount of data increases, the database must also grow, or scale. GridDB uses multiple nodes to provide scalability.

As the data set grows, additional nodes can be added to increase both database performance and increase the total amount of storage available. With GridDB Standard Edition, new nodes can be added without interruption while the cluster is online.

In a series of benchmark[4] tests performed by Fixstars, GridDB outperformed Apache Cassandra over the entire series of tests. The benchmarks used the YCSB on 1, 8, 16, and 32-node database clusters utilizing the Microsoft Azure Cloud Platform. GridDB showed that it scaled significantly better than Cassandra when new nodes were added.

4.3 Continuous Data

IoT sensors generate data 24 hours a day, 365 days per year. Downtime for either failure or maintenance is not an option like it is in applications that only require availability during business hours. After years of operation and trillions of transactions the database must remain as fast as it was immediately after installation.

To demonstrate how GridDB's architecture is best for workloads that cannot have interruption, Fixstars performed a 24-hour time-trial with GridDB and Cassandra that shows how GridDB is able to operate for extended periods with an update-intensive workload without maintenance unlike Cassandra and other log-sorted that require compaction and other maintenance.

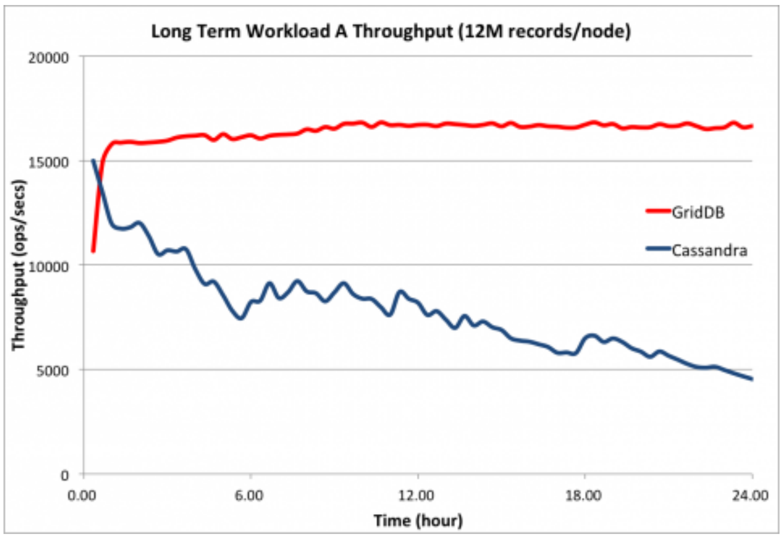


Figure 3: GridDB and Cassandra Performance Over Time

[4] https://www.griddb.net/en/docs/Fixstars_NoSQL_Benchmarks.pdf

The continuous ingestion of data also means that a database must be highly available and reliable which is accomplished through data replication on multiple nodes and fault tolerant algorithms to handle failover.

Distributed systems like GridDB systems typically use either a Master-Slave architecture or a Peer-to-Peer architecture for managing their nodes. The "Master" in a Master-Slave distributed system is typically a single point of failure and peer-to-peer systems, all nodes are identical but will incur some communication overhead to provide consistency. GridDB is a hybrid, any node is capable of being the master and in the event of a master-node failure, one of the followers will take over ensuring continuous service.

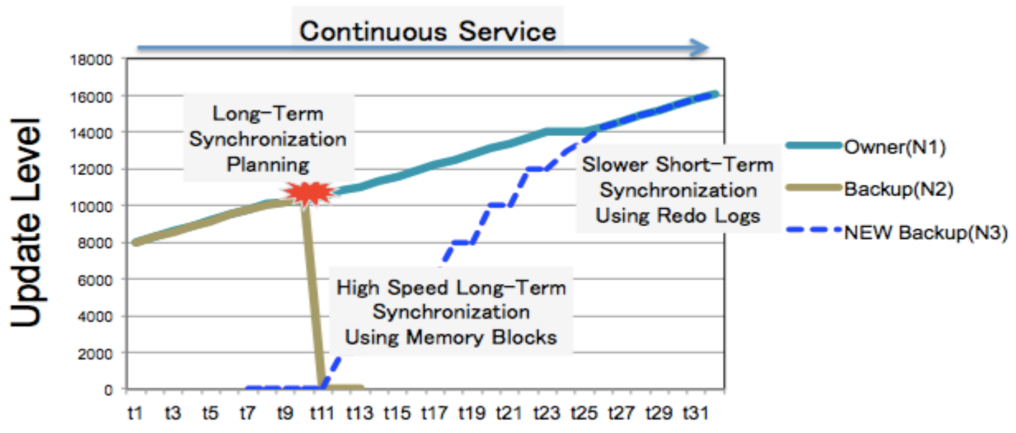


Figure 4: Autonomous Data Distribution Algorithm

In the event of a failure, the Autonomous Data Distribution Algorithm (ADDA) will re-assign owner or backup roles for a partition and instruct the new nodes to begin synchronization.

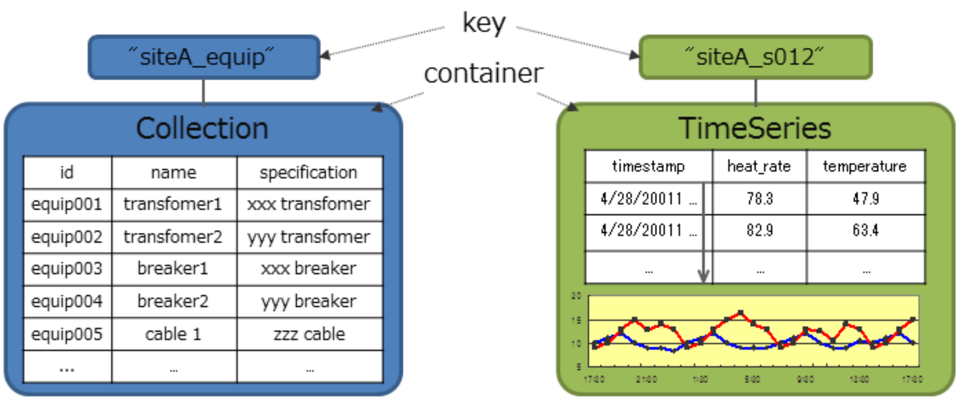
GridDB uses a hybrid architecture for cluster management. An algorithm is used to determine the master node and in the case of the master failing, a bully algorithm is run again to determine the new master node. Without being able to failover the master, the master becomes a single-point-of-failure (SPOF). This allows for GridDB to retain high reliability and the higher performance associated with master/slave architectures versus peer-to-peer architectures.

GridDB can be configured for either immediate or eventual consistency. With immediate consistency, the partition owner handles all read and write requests and propagates them to the backup nodes. In eventual consistency, replicas can respond to read requests.

4.4 Various Data Types

Applications regarding traffic, climate, and many other domains need reliable, constant sources of many different types of data including geospatial and temporal data to be effective. Many data objects used in IoT applications have spatial characteristics in multiple dimensions thus databases used for IoT applications should efficiently handle containers, schemas, and indexing of both temporal and spatial characteristics.

GridDB's key-container model of data allows for easy usage of varying types of data. The containers can use any key or a timestamp that allows for easier processing of temporal data. GridDB has built-in aggregation and geometry functions that enable developers to easily build queries without having to build their own complex routines to perform the same functions.



id	name	specification
equip001	transfomer1	xxx transfomer
equip002	transfomer2	yyy transfomer
equip003	breaker1	xxx breaker
equip004	breaker2	yyy breaker
equip005	cable 1	zzz cable
...	...	...

timestamp	heat_rate	temperature
4/28/20011 ...	78.3	47.9
4/28/20011 ...	82.9	63.4
...	...	...

2 types of container, collection and time series
A container is like the RDB table of a row/column

Figure 5 TimeSeries and Collection Containers

Containers can either be a Collection or a TimeSeries; a collection can use any type of data for a key while a TimeSeries Container uses a time stamp for a key.

TimeSeries Containers allow for special time functions in dealing with time-stamped data. Timestamps can be used to delete certain data after a set amount of time has passed. TimeSeries containers also support compression allowing more efficient storage of archived data.

GridDB's time-specific queries and functions include time-weighted averages as well as the ability to perform linear interpolation to estimate data values. There is also the ability to set consistent sampling periods setting start and end times and a set time interval between returned values.

Time Query Example:

```
SELECT TIME_SAMPLING(voltage103, TIMESTAMP('2011-07-
01T00:00:00Z'), TIMESTAMP('2011-07-02T00:00:00Z'), 1,
HOUR) FROM plant1
```

GridDB SE (Standard Edition) and above can support spatial data as column types for its containers along with geometric queries. For Geometric data, objects can be created through the C and Java APIs or through TQL queries. GridDB accepts objects in WKT (well-known-text) form and supports objects like POINT, POLYGON, LINESTRING, POLYHEDRALSURFACE, and QUADRICSURFACE. GridDB also offers the use of several ST_ functions in TQL queries like intersections to be performed on Geometric data.

Creating a Geometry Object with the Java API:

```
    Geometry coordinate = Geometry.valueOf("POINT(33.651442 -
    117.744744)");
```

Geometry objects can be created with TQL through ST_GeomFromText function. Other objects such as rectangles, planes, spheres, cones, and cylinders can also be created with TQL. GIS functions such as generating SRIDBEs and calculating intersections between geometric objects are also supported.

The following TQL example returns results with points within the given polygon:

```
    SELECT * WHERE ST_MBRIntersects (geom, ST_GeomFromText
('POLYGON ((0 0,10 0,10 10,0 10,0 0))'))
```

The unique key-container data model used by GridDB has the benefit of providing ACID characteristics that can be guaranteed at the container level. In this model, a KEY can represent one specific sensor out in the field while the VALUE (CONTAINER) can represent all the data incoming from that sensor. The CONTAINER mostly resembles a traditional relational table with columns and rows. Data access uses the key to narrow down and find rows and containers. This type of data access allows temporal, spatial and other kinds of data to be processed quickly.

4.5 Real Time Processing

IoT applications also require analysis while data is being ingested, such as with high-speed search and pattern recognition.

Stream Processing allows applications to collect, integrate, and visualize real-time stream data. This means applications can process and act on their data as soon as it is produced, meaning data can be seen as infinite streams. These types of queries allow analysis on large amounts of data from multiple sources in real-time. This form of processing allows for businesses to adapt and conform to their analytical and business needs at a faster pace.

In the context of IoT, a well-developed application that utilizes real-time stream processing can solve many different challenges. TQL combined with Key-Container multi-get queries can be used to perform high-speed searches that will detect anomalies and abnormalities to provide quick responsiveness. Stream processing also allows for live monitoring as well as for automated alerts and notifications.

4.6 Batch Processing

Batch processing is defined as the processing of a group or "batch" of transactions at once. No user interaction should be required. Batch or more transactional processing is used to help automate actions and decisions in IoT such as generating reports for a certain period like for monthly billing.

Batch processing can be cheaper and more efficient than transactional processing and allows businesses and organizations to carry out large tasks during off the clock periods where the strain on resources is smaller.

One of the most popular frameworks used in batch processing is Apache Hadoop which has a layer known as MapReduce. MapReduce is Hadoop's native batch processing engine. This engine allows effective and inexpensive processing of large data sets when time is not a large factor. GridDB supports batch processing with a connector to MapReduce and the Hadoop File system.

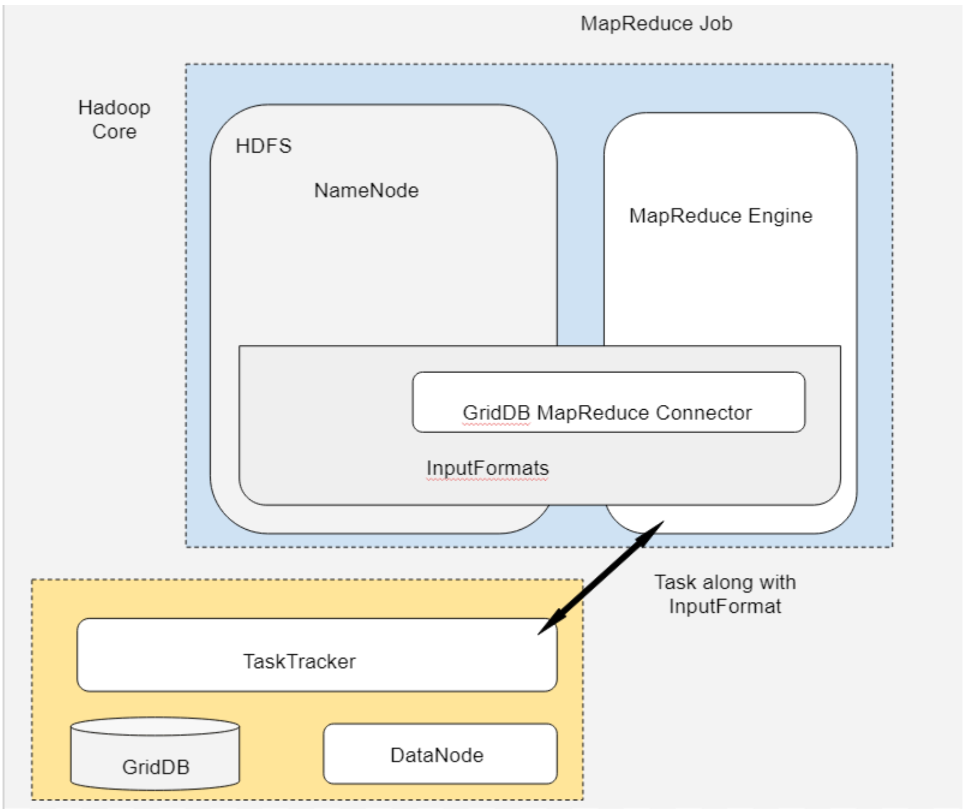


Figure 6: GridDB as part of a MapReduce Application.

GridDB databases can be used as input sources as well as output destinations for MapReduce batch jobs. With GridDB as the storage engine and MapReduce as the processing engine, a foundation is provided to process multiple workloads at a time from many different domains. This connector, when put in combination with GridDB's high performance from parallel and in-memory processing, allow MapReduce to handle more diverse workloads.

4.7 Ad-hoc User Queries

Ad-hoc or user queries are created spontaneously whenever the need to get certain information arises and may involve adjusting 'WHERE' clauses or other location or source specific conditions. For example, depending on the choice a user makes in a user-interface, it may change what values in the WHERE clause are set or which containers the database selects from.

GridDB provides ad-hoc queries through TQL. TQL is a simplified version of SQL for NoSQL products. Ad-hoc queries can be made through a Query object in GridDB's APIs. Queries can be generated simply by using TQL strings. These queries can be generic selection queries as well as time-specific aggregations and geometry queries and operations. Those strings contain the keywords, ranges, options, and sources for the query. TQL has the benefit of being created dynamically and can be adjusted depending on the application's context. Once these strings are made, the query object is created and later fetched.

TQL Example (Java API):

```
     Query<Row> query = collection.query("SELECT * FROM
sensors WHERE volts = '" + voltage + "'");
```

GridDB also has a functional Apache Spark with a database connector. Apache Spark is a parallel data processing framework to provide fast data analytics. Using the connector allows a GridDB database to be used as an input source for Spark queries and analytics. Its interactive shell can be used to quickly and easily perform ad-hoc queries by data scientists or developers or can be built into user-facing business applications.

5 Use Cases

GridDB has been already implemented in several different IoT projects:

An industrial manufacturing company selected GridDB as their database for their global compressor management system. The system provided cloud services to collect, store, analyze, and visualize data from compressors from around the globe. Processing data at this scale allows for comprehensive support and maintenance packages worldwide.

In 2015, Toshiba began offering the Building Energy Management Systems (BEMS) with GridDB as its database. BEMS monitors and controls a building's needs, which include heating, ventilation, air conditioning, lighting, and security. GridDB stored 2TB of data from hundreds of buildings with thousands of records being transacted each second.

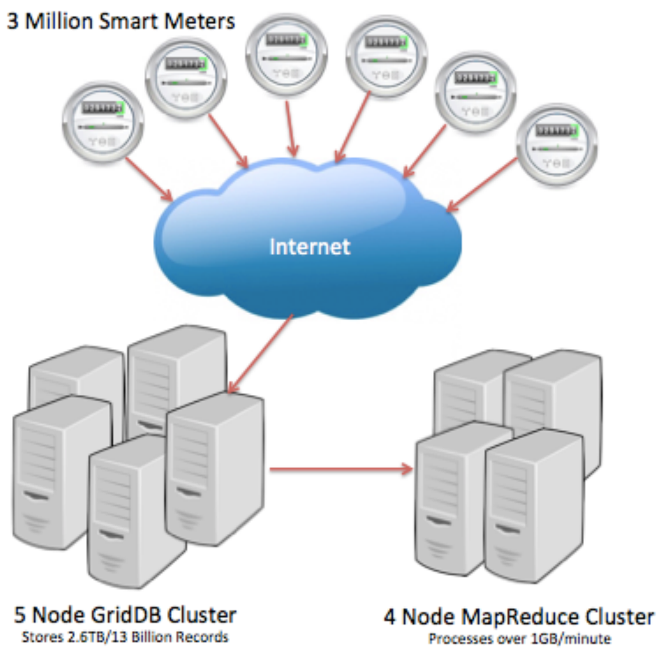


Figure 7: GridDB Solution to process Smart Meter data.

In 2016, GridDB was used to collect data from distributions of smart meters for an electric power company in Japan. The company had a total of 3 million smart meters. Smart meter data would be collected every 30 minutes and stored for 3 months. This resulted in a data set over 13 billion records totaling 2.6TB. Thanks to the GridDB's fast performance as well as its support for Hadoop MapReduce processing, it takes 40 minutes to analyze 43.2GB of data. This rate improves performance by over 2000 times when compared to the previous implementation.

6 Conclusion

GridDB provides high throughput with an in-memory and persistent storage and a fast hybrid master/slave cluster management model. With modifiable containers, GridDB provides flexibility that is able to easily adapt as your data changes. Your data is safe in GridDB with its reliability and consistency features. GridDB is one of only a few ACID - compliant NoSQL databases and its Autonomous Data Distribution Algorithm (ADDA) algorithm ensures data is efficiently replicated in the case of a failure.

Meanwhile, developing applications is easy with a SQL-like TQL query language and native Java, Python, Ruby, and C APIs. Working with real world data is made easy using the geometry functions and TimeSeries container. GridDB also offers the ability to integrate with other open source projects such as Kafka, Hadoop MapReduce, Apache Spark, and KairosDB.

While selecting a database for an IoT application depends on particular project needs, GridDB is an ideal choice for most workloads because it provides high performance along with the flexibility and reliability required over the lifetime of the project.