TOSHIBA
Leading Innovation

Building IoT Applications with GridDB

August 23, 2017
Revision 0.3

GridDB™

Table of Contents

Executive Summary 2
Introduction 2
  Components of an IoT Application 2
IoT Models 3
  Edge Computing 4
  Gateway Aggregation 4
  Direct Connectivity 5
Requirements for IoT Databases 6
  Scalability 6
  Flexibility 7
    Key-Container Model 8
    Geospatial and Temporal Data Types 9
  Reliability 10
    Failover Support 10
    Replication System 10
    Hybrid Cluster Management 11
Effective Data Analysis 11
  Stream Processing 11
  Batch Processing 12
  Ad-hoc User Queries 13
Use Cases 14
Conclusion 15

List of Figures

Figure 1: Visualization of IoT Models 4
Figure 2: Difference in Scalability between GridDB and Cassandra 6
Figure 3: GridDB and Cassandra Performance Over Time 7
Figure 4: GridDB's Collection and TimeSeries Container Types 8
Figure 5: Autonomous Data Distribution Algorithm 11
Figure 6: GridDB as part of a Map Reduce Application 13
Figure 7: GridDB Solution for Smart Meter Data 15

Executive Summary

This white paper examines GridDB as an IoT (Internet of Things) database. It details how GridDB's architecture and overall structure accommodates and complements the key aspects needed by an IoT database for optimal performance. This document also explains how GridDB supplies the necessary properties needed in large scale data management. GridDB's other features, in terms of API support and connectors to other software frameworks, help give IoT applications more extendibility.

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.

Components of an IoT Application

IoT applications can be broken down into three general components: sensors, communications, and intelligent systems.

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.

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

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 an internet cloud service 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.

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.

The above three components combined create the IoT application which provides services to the user such as reporting, analytics, and remote control. Common applications in IoT include smart homes, smart cities, and smart healthcare.

IoT Models

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.

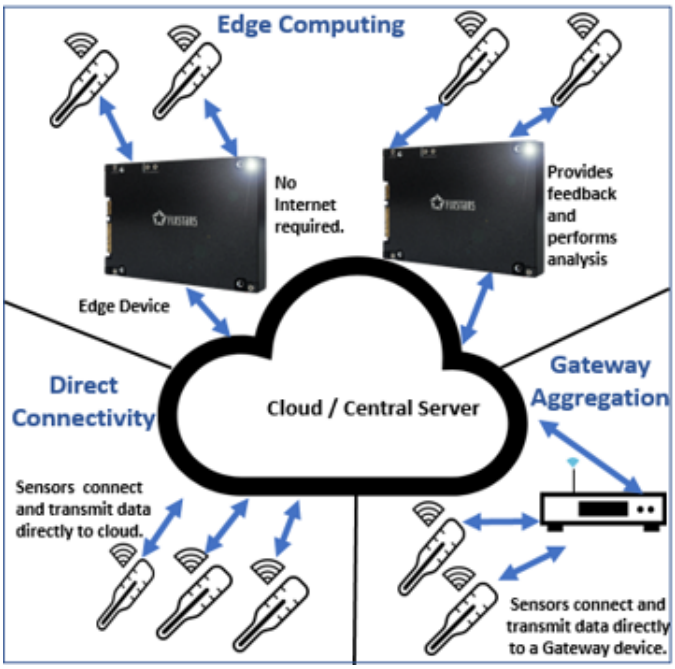


Figure 1: Visualization of IoT Models

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 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.

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

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

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.

GridDB can be used in a gateway device through its APIs or the gateways can use a secondary messaging protocol such as MQTT to forward the data to the data center or cloud where the data would be ingested.

Direct Connectivity

IoT sensors and devices can directly send their data directly 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 or messages can be sent via an application gateway in the centralized infrastructure such as an MQTT or Kafka server.

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

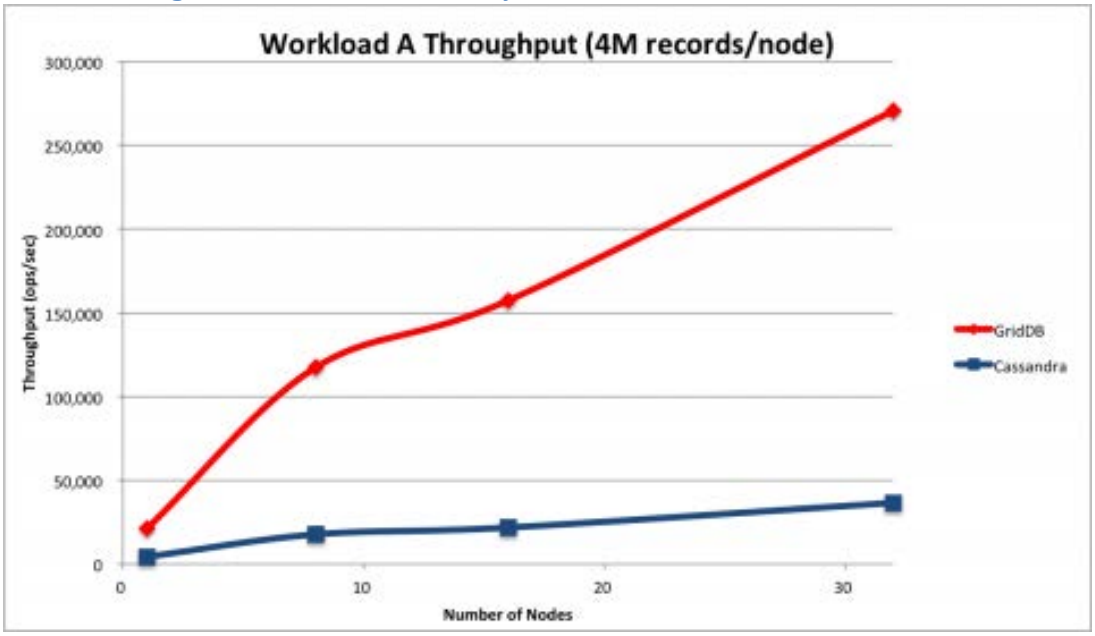
Requirements for IoT Databases

The specific needs and requirements for an IoT database are heavily dependent on the application itself, but in general[4] an IOT database needs to be scalable, reliable, and flexible.

Scalability

At its very core, scalability is the database's ability to meet growing demands as requirements and expectations grow. A "scalable" system should respond by utilizing newly added nodes to enhance performance and other criteria. The rapid growth of IoT means that more sensors and devices are being used to output and process data at larger scales; having a scalable database is becoming more and more important for IoT development.

Figure 2: Difference in Scalability between GridDB and Cassandra



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.

GridDB uses in-memory processing and storage as a way to handle high velocity data. It aims to keep most or all of its data in-memory, using checkpoint intervals to flush its internal memory structure back to disk. GridDB uses its Last Recently Used (LRU) and data affinity algorithms to determine which data records stay in-memory. Affinity functions make effective use and operation of limited memory areas in a database.

[4] http://www.ijcaonline.org/archives/volume159/number8/gurav-2017-ijca-913021.pdf

GridDB uses multiple nodes to provide scalability to the database. Large data sets are distributed across multiple nodes through partitioning and replication. By keeping an entire container in a single partition, performance is further improved by removing the need for internode coordination. GridDB automatically balances the cluster ensuring each node is assigned a similar number of partitions.

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[5] 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. The 24 hour time-trial shows that GridDB is able to operate for extended periods with an update-intensive workload without maintenance unlike the log-sorted databases (such as Cassandra) that require compaction and other maintenance.

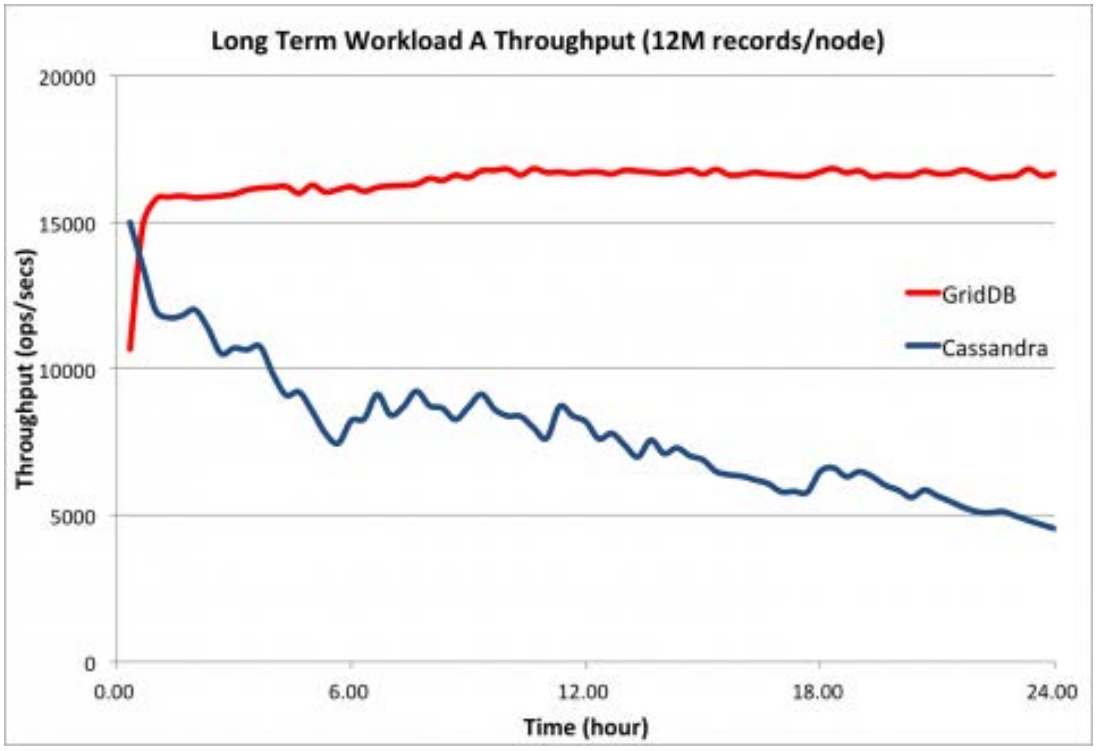


Figure 3: GridDB and Cassandra Performance Over Time

Flexibility

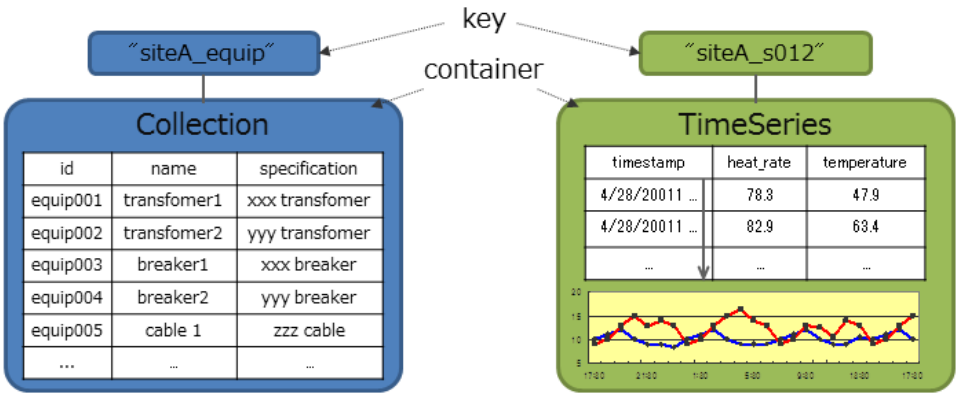
[5] https://www.griddb.net/en/docs/Fixstars_NoSQL_Benchmarks.pdf

Flexibility means accepting the variety of data types likely to appear over the life of the solution. A NoSQL approach provides the flexibility needed to fit different data models to changing application requirements. Another factor to consider is the high velocity to which data is produced. Often this means that the structure of data is not known ahead of time, meaning the database must be able to cope with these occurrences.

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. The schema of a container can also be modified to evolve over time allowing columns to be both removed and added. Furthermore, GridDB has built-in aggregation and geometry functions that enable developers to easily queries without having to build their own complex routines to perform the same functions.

Key-Container Model

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 many different kinds of data to be processed quickly.



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

Figure 4: GridDB's Collection and TimeSeries Container Types

The Key-Container model offers the choice of making the Container either a "collection" container, accepting any type of key data, or a TimeSeries Container which uses a TimeStamp for a key.

The containers in GridDB also have the benefit having an updatable schema. Column can be added or removed on a per-row basis and different types of row-keys and indexes can be added and removed from existing containers.

To help keep datasets to a reasonable size over time, compression is also supported as well as the automatic deletion of data over a certain age.

Geospatial and Temporal Data Types

IoT systems frequently gather data regarding temporal and geospatial characteristics. Applications regarding traffic, climate, and many other domains need reliable, constant sources of geospatial and temporal data to be effective. Data objects used in IoT applications have spatial characteristics in multiple dimensions. Databases used for IoT should support containers, schemas, and indexing of both temporal and spatial characteristics.

GridDB's TimeSeries container uses the timestamp as the row-key. Out-of-the-box acceptance and utilization of time data lends a rather large advantage to GridDB in this area.

These 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 data expiration and compression for providing effective data management.

GridDB also supports several time-specific queries and functions as well. It supports the use of time-weighted averages as well as the ability to perform linear interpolation to estimate data values. There is also the ability to get a time-sampling with start and end time and a set time interval between 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 API's 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 SRID's 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))'))
```

Reliability

Reliability means high-availability and peace of mind; it is having the ability to stay online and accept requests even if multiple nodes experience sudden failure.

GridDB has multiple mechanisms and features to provide availability, reliability, and consistency. By storing data on multiple nodes using replication, it is able to withstand node failure of both its master and follower nodes, providing both the performance of a Master-Slave architecture and the reliability of a peer-to-peer architecture.

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.

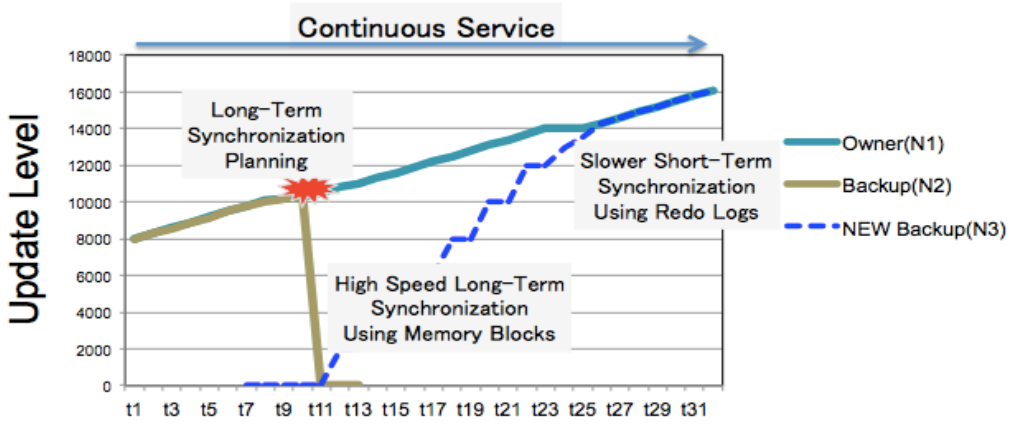
Failover Support

GridDB's method of handling node failure is by essentially having three "tiers" of nodes. A node can either be a partition owner, backup node, or a catch-up. GridDB's master node monitors its slave nodes by periodically checking for a response to its "heartbeat." When a slave node goes down it is marked and its partitions are assigned to other replicas. When a master fails, a backup node is "promoted" to master, and a catch-up node is "promoted" to take that backup node's place. This entire process happens automatically.

Replication System

GridDB allows for the custom setting of replication levels for an entire cluster of nodes. Every partition has a node that is its owner and then the specified number of replicas or backup nodes. 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.

Figure 5: Autonomous Data Distribution Algorithm



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.

Hybrid Cluster Management

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.

Effective Data Analysis

One key part of making an IoT application scalable and effective is the use of data analytics. IoT devices create massive amounts of data from a variety of contexts. Using data analytics, whether performed on real-time data or historical data, can provide invaluable insights for both the organization and the users. In general, data processing in IoT can be broken down into three approaches: real-time or stream processing, batch jobs, and ad-hoc user queries.

Stream Processing

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. Real-time stream processing can aid in faster detection of anomalies and abnormalities to provide quick responsiveness. Stream processing also allows for live monitoring as well as for automated alerts and notifications.

Many applications that utilize stream processing use messaging brokers such as Apache Kafka and MQTT. MQTT provides lightweight machine-to-machine communication and can be used on nearly any IoT device. It uses a publish-subscribe model to send messages across devices and different services through its API. GridDB provides connectors to Kafka and MQTT so database querying and storage can be integrated seamlessly for an IoT application.

Kafka is another messaging broker that uses the publish-subscribe model. It streams data to provide real-time streaming and data pipelines. GridDB provides connectors to Kafka and MQTT through their APIs. MQTT gives efficient and fast communication between IoT sensors. Kafka gives real-time data messaging to provide real-time processing. Using GridDB in combination with MQTT and Kafka services gives an IoT application quick and efficient data processing.

Batch Processing

Batch processing is defined as the processing of a group or "batch" of transactions at once. Once the processing begins, no user interaction should be required. Batch or more transactional processing is used to help automate actions and decisions in IoT. Batch processing provides personalized data so that an application can respond to certain events.

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. It is often used for creating analytics or reports at specific time intervals such as a monthly report that uses the previous month's worth of records at once without intervention.

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.

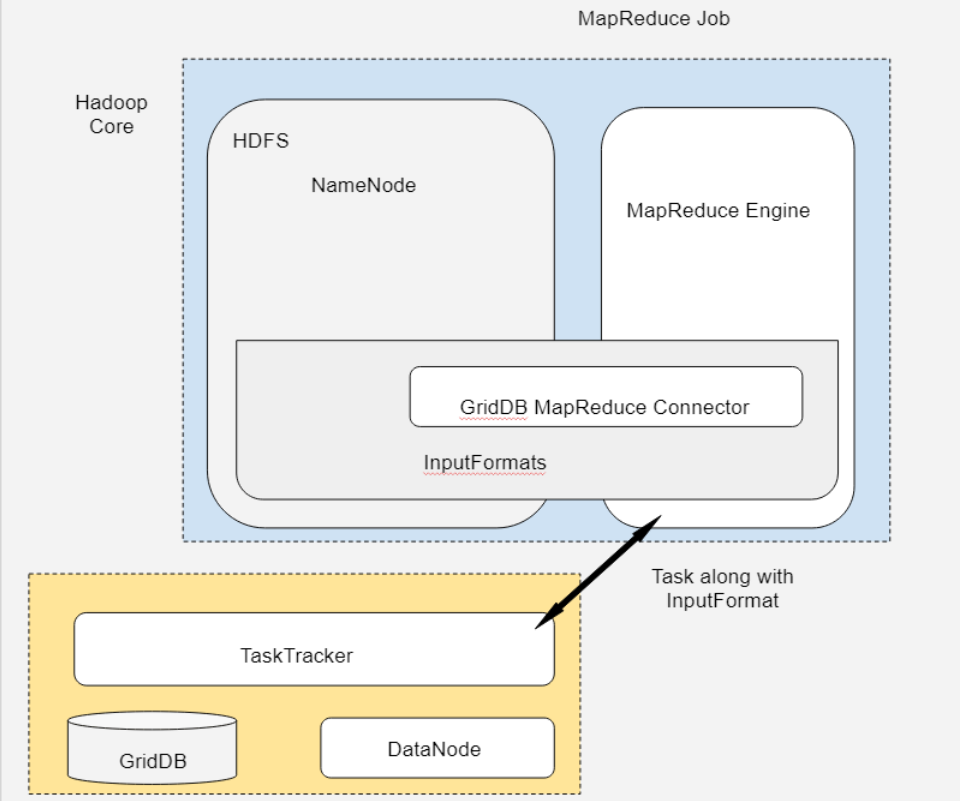


Figure 6: GridDB as part of a Map Reduce Application.

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.

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.

Use Cases

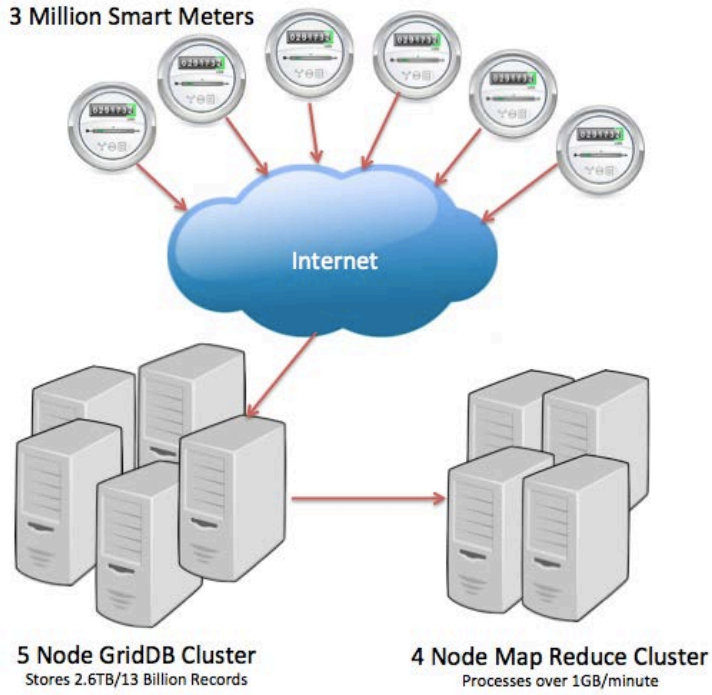
GridDB has been already implemented in several different IoT projects:

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 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.

Figure 7: GridDB Solution for Smart Meter Data



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

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.