GridDB Technical Reference

Revision: 1040

Table of Contents

1. Introduction

1.1 Aim & configuration of this manual

This manual explains the GridDB architecture and functions provided.

This manual is targeted at administrators who are in-charge of the operational management of GridDB and designers and developers who perform system design and development using GridDB

The manual is composed as follows.

- What is GridDB?
  - Describes the features and application examples of GridDB.
- Architecture of GridDB
  - Describes the data model and cluster operating structure in GridDB.
- Functions provided by GridDB
  - Describes the data management functions, functions specific to the data model and operating functions provided by GridDB.
- Parameters
  - Describes the parameters to control the operations in GridDB.

2 What is GridDB?

GridDB is a distributed NoSQL database to manage a group of data (known as a row) that is made up of a key and multiple values. Besides having a composition of an in-memory database that arranges all the data in the memory, it can also adopt a hybrid composition combining the use of a disk (including SSD as well) and a memory. By employing a hybrid composition, it can also be used in small scale, small memory systems.

In addition to the 3 Vs (volume, variety, velocity) required in big data solutions, data reliability/availability is also assured in GridDB. Using the autonomous node monitoring and load balancing functions, laborsaving can also be realized in cluster applications.

2.1 Features of GridDB

2.1.1 Big data (volume)

As the scale of a system expands, the data volume handled increases and thus the system needs to be expanded so as to quickly process the big data.

System expansion can be broadly divided into 2 approaches - scale-up (vertical scalability) and scale-out (horizontal scalability).

- What is scale-up?

  This approach reinforces the system by adding memory to the operating machines, using SSD for the disks, adding processors, and so on. Generally, there is a need to stop the nodes once during scale-up operation as it is not a cluster application using multiple machines even though each individual processing time is shortened and the system processing speed is increased. When a failure occurs, failure recovery is also time-consuming.

- What is scale-out?

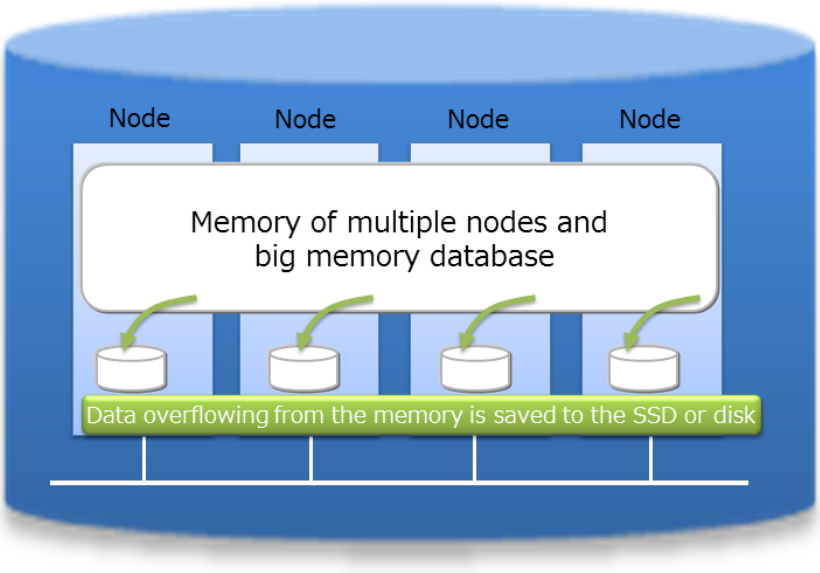
  This approach increases the number of nodes constituting a system to improve the processing capability. Generally, there is no need to completely stop service when a failure occurs and during maintenance as multiple nodes are linked and operating together. However, the application management time and effort increases as the number

of nodes increases. This architecture is suitable for performing highly parallel processing.

In GridDB, in addition to the scale-up approach to increase the number of operating nodes and reinforce the system, new nodes can be added to expand the system with a scale-out approach to incorporate nodes into an operating cluster.

As an in-memory processing database, GridDB can handle a large volume of data with its scale-out model. In GridDB, data is distributed throughout the nodes inside a cluster that is composed of multiple nodes. Therefore, a large-scale memory database can be provided as the memories of multiple nodes can be used as a single, large memory space.

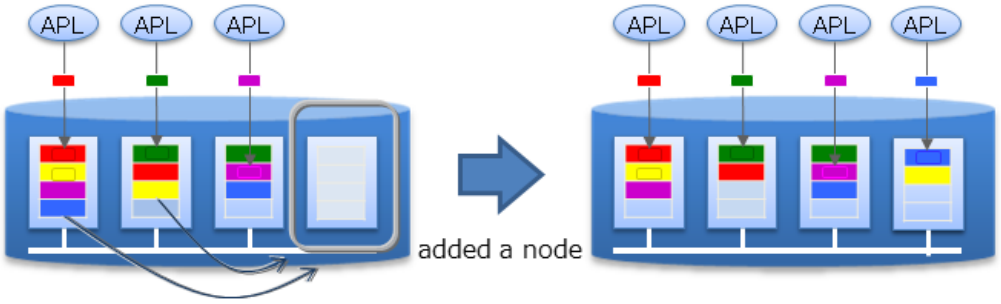
In addition, since data management of a hybrid composition that combines the use of disk with memory is also possible, data exceeding the memory size can be retained and accessed even when operating with a standalone node. A large capacity that is not limited by the memory size can also be realized.



Combined use of in-memory/disk

System expansion can be carried out online with a scale-out approach. As a result, a system in operation can be supported without having to stop it as it will support the increasing volume of data as the system grows.

In the scale-out approach, data is arranged in an appropriate manner according to the load of the system in the nodes built into the system. As GridDB will optimize the load balance, the application administrator does not need to worry about the data arrangement. Operation is also easy because a structure to automate such operations has been built into the system.

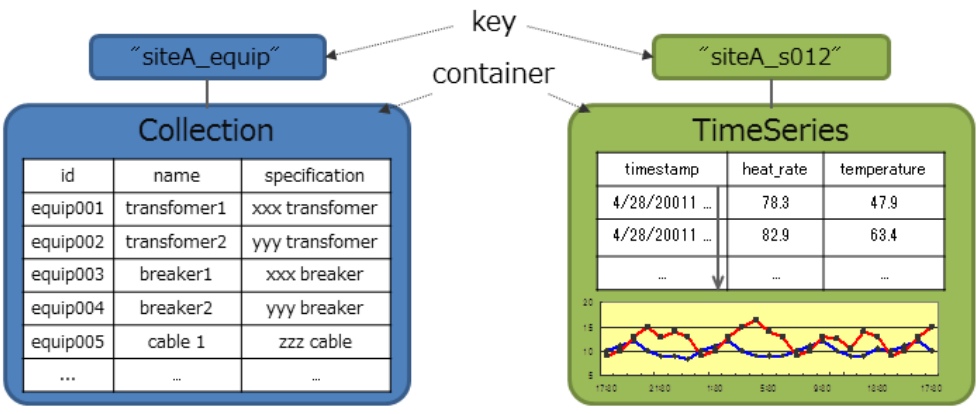


Scale-out model

2.1.2 Various data types (variety)

GridDB data adopts a Key-Container data model that is expanded from Key-Value. Data is stored in a device equivalent to a RDB table known as a container. (A container can be considered a RDB table for easier understanding.)

When accessing data in GridDB, the model allows data to be short-listed with a key thanks to its Key-Value database structure, allowing processing to be carried out at the highest speed. A design that prepares a container serving as a key is required to support the entity under management.



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

Besides being suitable for handling a large volume of time series data (TimeSeries container) that is generated by a sensor or the like and other values paired with the time of occurrence, space data such as position information, etc. can also be registered and space specific operations (space intersection) can also be carried out in a container. A variety of data can be handled as the system supports non-standard data such as array data, BLOB and other data as well.

A unique compression function and a function to release data that has expired and so on are provided in a TimeSeries container, making it suitable for the management of data which is generated in large volumes.

2.1.3 High-speed processing (velocity)

A variety of architectural features is embedded in GridDB to achieve high-speed processing.

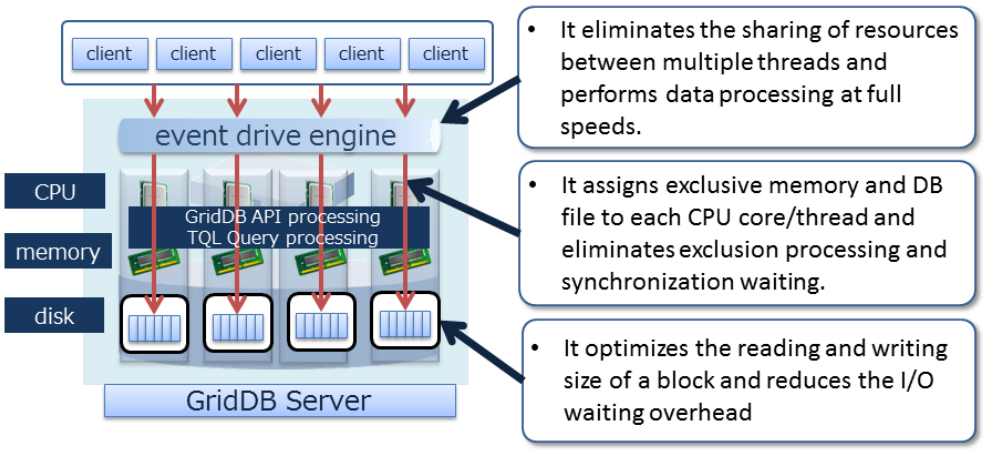
- Processing is carried out in the memory space as much as possible

  In the case of an operating system with an in-memory in which all the data is arranged, there is no real need to be concerned about the access overhead in the disk. However, in order to process a volume of data so large that it cannot be saved in the memory, there is a need to localize the data accessed by the application and to reduce access to the data arranged in the disk as much as possible.

  In order to localize data access in GridDB, a function is provided to arrange related data in the same block as far as possible. Since data in the data block can be consolidated according to the hints provided in the data, memory mishit is reduced during data access, thereby increasing the processing speed for data access. By setting hints for memory consolidation according to the access frequency and access pattern in the application, limited memory space can be used effectively for operation. (Affinity function)

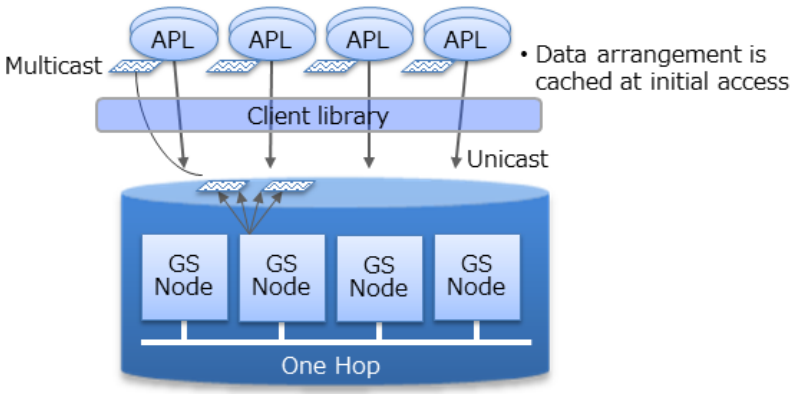
- Reduces the overhead

  In order to reduce events that cause delay in the database execution by as much as possible e.g. a lock or latch event when accessing the database in parallel, exclusive memory and DB files are assigned to each CPU core and thread, so as to eliminate time spent waiting for exclusion and synchronization processing to be carried out.



Architecture

In addition, direct access between the client and node is possible in GridDB by caching the data arrangement when accessing the database for the first time on the client library end. Since direct access to the target data is possible without going through the master node to manage the operating status of the cluster and data arrangement, access to the master node can be centralized to reduce communication cost substantially.



Access from a client

- Processing in parallel

High-speed processing is realized through parallel processing e.g. by dividing a request into processing units capable of parallel processing in the drive engine and executing the process using a thread in the node and between nodes, as well as dispersing a single large data into multiple nodes (partitioning) for processing to be carried out in parallel between nodes.

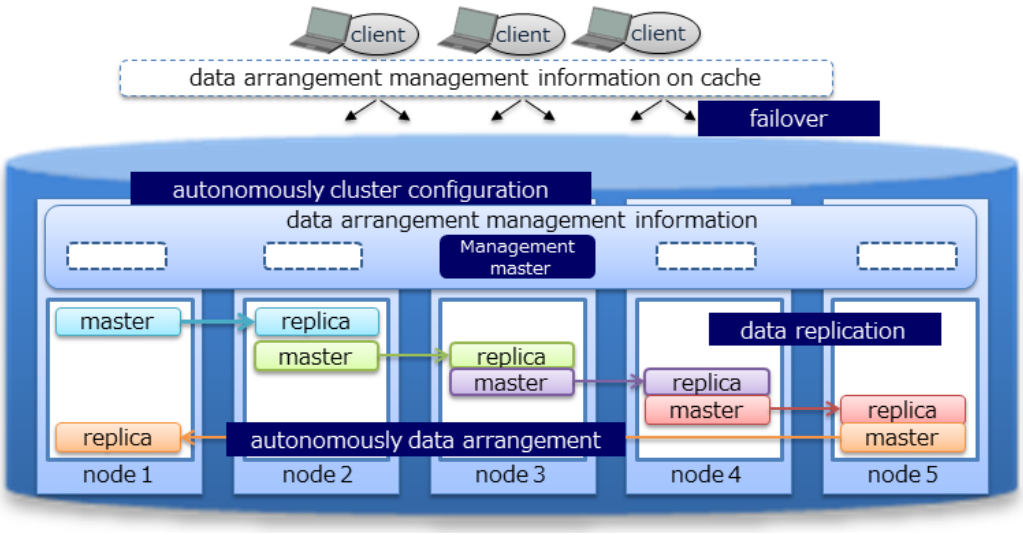
2.1.4 Reliability/availability

Duplicate data (hereinafter known as replicas) are created in the cluster and processing can be continued by using these replicas even when a failure occurs in any of the nodes constituting a cluster. Special operating procedures are not necessary as the system will also automatically perform re-arrangement of the data after a node failure occurs (autonomous data arrangement). Data arranged in a failed node is restored from a replica and then the data is re-arranged so that the set number of replicas is reached automatically.

Duplex, triplex or multiplex replica can be set according to the availability requirements.

Each node performs persistence of the data update information using a disk, and all registered and updated data up to that point in time can be restored without being lost even if a failure occurs in the entire cluster system.

In addition, since the client also possesses cache information on the data arrangement and management, upon detecting a node failure, it will automatically perform a failover and data access can be continued using a replica.



High availability

3 Structure of GridDB

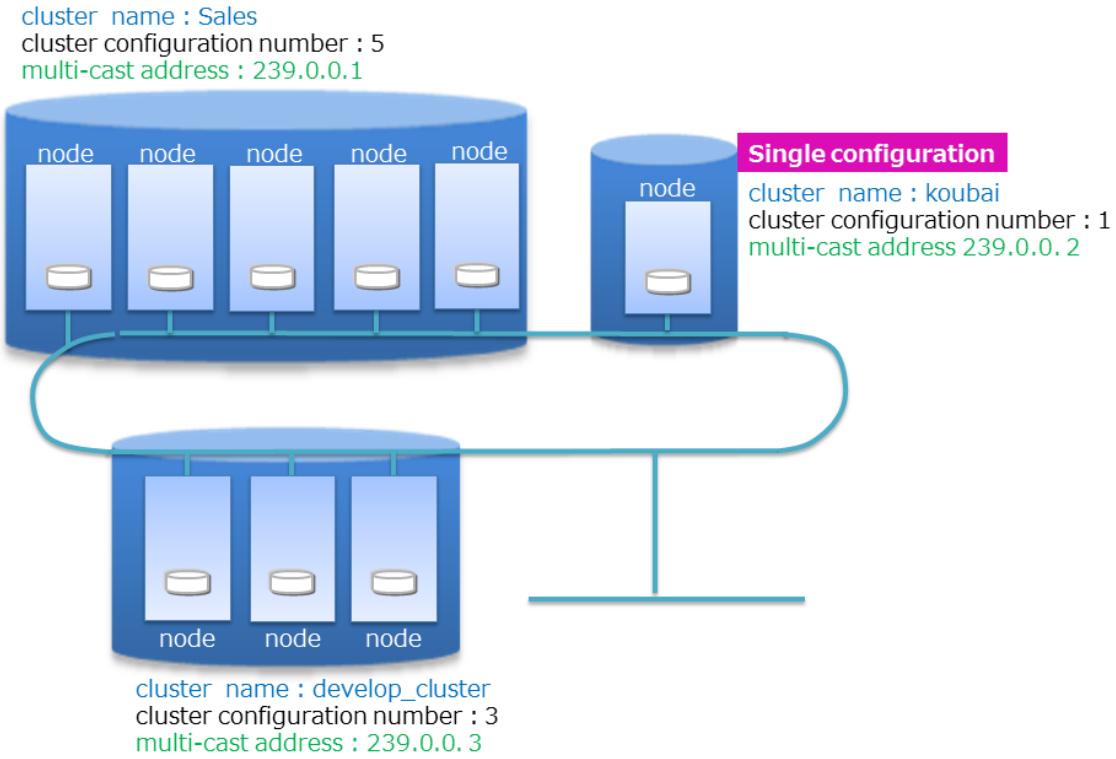
The operating structure and data model of a GridDB cluster is described.

3.1 Composition of a cluster

GridDB is operated by clusters which are composed of multiple nodes. To access the database from an application system, the nodes have to be started up and the cluster has to be constituted (cluster service is executed).

A cluster is formed and cluster service is started when a number of nodes specified by the user joins the cluster. Cluster service will not be started and access from the application will not be possible until all nodes constituting a cluster have joined the cluster.

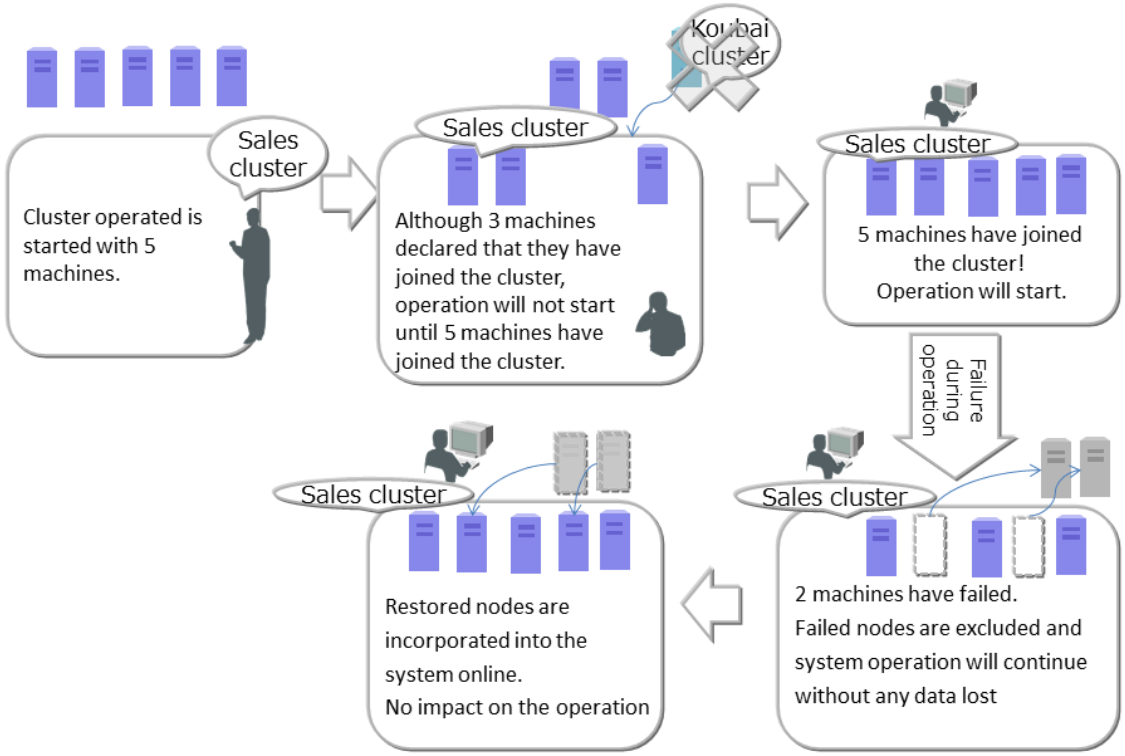
A cluster needs to be composed even when operating with 1 node only. In this case, the number of nodes constituting a cluster is 1. A composition that operates a single node is known as a single composition.



Cluster name and number of nodes constituting a cluster

Cluster names are used to separate multiple clusters so that the correct clusters (using the intended nodes) can be composed using multiple GridDB nodes on a network. Multiple GridDB clusters can be composed in the same network. A cluster is composed of nodes with the same cluster name, number of nodes constituting a cluster, multi-cast address setting. When composing a cluster, the parameters need to be specified as well in addition to setting the cluster name in the cluster definition file which is a definition file saved for each node constituting a cluster.

The operation of a cluster composition is shown below.



Operation of a cluster composition

To start up a node and compose a cluster, the operation commands gs_startnode/gs_joincluster command or gs_sh are used. In addition, there is a service control function to start up the nodes at the same time as the OS and to compose the cluster.

To compose a cluster, the number of nodes joining a cluster (number of nodes constituting a cluster) and the cluster name must be the same for all the nodes joining the cluster.

Even if a node fails and is separated from the cluster after operation in the cluster started, cluster service will continue so long as the majority of the number of nodes is joining the cluster.

Since cluster operation will continue as long as the majority of the number of nodes is in operation, when a node is separated online due to maintenance and other work during cluster operation, it can be incorporated after the maintenance work ends. Furthermore, nodes can be added online to reinforce the system.

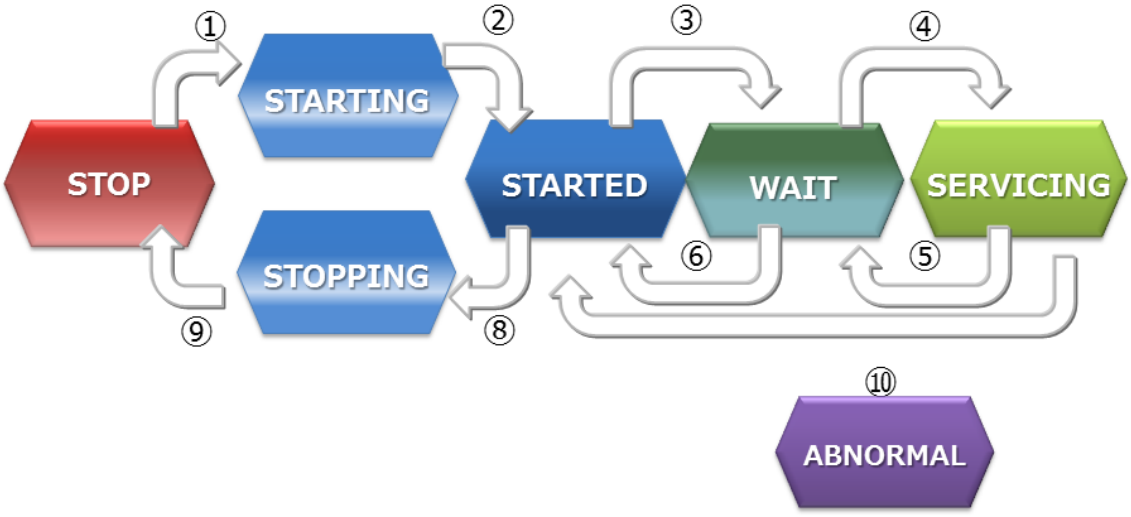
3.1.1 Status of node

There are 2 GridDB status, nodeStatus and clusterStatus that can be checked with a gs_stat command. The status of a node is determined by these 2 statuses.

nodeStatus indicates the operating status of the node while clusterStatus indicates the role of each node in the constituted cluster. The status of the entire cluster is determined by the status of these multiple nodes belonging to the cluster.

- Transition in the node status

  The node status may be one of the following statuses shown in the diagram below.



Node status

  - [STOP]: State in which the GridDB server has not been started in the node.
  - [STARTING]: State in which the GridDB server is starting in the node.
    Depending on the previous operating state, start-up processes such as recovery

processing of the database are carried out. The only possible access from a client is checking the status of the system with a gs_stat command or gs_sh command. Access from the application is not possible.

- [STARTED]: State in which the GridDB server has been started in the node. However, continued access from the application is not possible as the node has not joined the cluster. To obtain the cluster composition, a command is issued to join a cluster with the gs_joincluster or gs_sh cluster operating command.
- [WAIT]: State in which the system is waiting for the cluster composition. Nodes have been informed to join a cluster but the number of nodes constituting a cluster is insufficient, so the system is waiting for the number of nodes constituting a cluster to be reached. It also indicates the node status when the number of nodes constituting a cluster drops below the majority and the cluster service is stopped.
- [SERVICING]: State in which a cluster has been constituted and access from the application is possible. However, access may be delayed if synchronization between the clusters of the partition occurs due to a re-start after a failure when the node is stopped or the like.
- [STOPPING]: Intermediate state in which a node has been instructed to stop but has not stopped yet.
- [ABNORMAL]: SERVICING state or state in which an error is detected by the node in the middle of the state transition. A node in the ABNORMAL state will be automatically separated from the cluster. After obtaining the operating information of the system, the system needs to be stopped by force and then re-started. By re-starting the system, recovery processing will be automatically carried out.

- Description of state transition: A description of events that serve as an opportunity to change the status of a node.

State transition	State transition event	Description
①	Command execution	Node start-up using gs_startnode command, gs_sh, service start-up
②	System	Automatic transition at the end of recovery processing or loading of database files

State transition	State transition event	Description
③	Command execution	Cluster participation using gs_joincluster/gs_appendcluster command, gs_sh, service start-up
④	System	State changes when the required number of component nodes join a cluster
⑤	System	When other nodes that make up a cluster are detached from the service due to a failure, etc., and the number of nodes constituting a cluster drops below half of the value set.
⑥	Command execution	Detaches a node from a cluster using a gs_leavecluster command or gs_sh
⑦	Command execution	Detaches a node from a cluster using a gs_leavecluster/gs_stopcluster command or gs-sh
⑧	Command execution	Stops a node using gs_stopnode command, gs_sh, service stop
⑨	System	Stops the server process once the final processing ends
⑩	System	Detached state due to a system failure. In this state, the node needs to be stopped by force once.

- Node status and nodeStatus, clusterStatus

  By using a gs_stat command, the detailed operating information of the node can be checked with text in the json format. The relationship between the clusterStatus and the nodeStatus which is a json parameter to indicate the gs_stat is shown below.

Status	/cluster/nodeStatus	/cluster/clusterStatus
STARTING	INACTIVE	SUB_CLUSTER
STARTED	INACTIVE	SUB_CLUSTER

Status	/cluster/nodeStatus	/cluster/clusterStatus
WAIT	ACTIVATING or DEACTIVATING	SUB_CLUSTER
SERVICING	ACTIVE	MASTER or FOLLOWER
STOPPING	NORMAL_SHUTDOWN	SUB_CLUSTER
ABNORMAL	ABNORMAL	SUB_CLUSTER

The status of the node can be checked with gs_sh or gs_admin.

3.1.2 Status of cluster

The cluster operating status is determined by the state of each node, and the status may be one of 3 states - IN OPERATION/INTERRUPTED/STOPPED.

Cluster service starts when all the nodes that make up a cluster (number of nodes constituting a cluster) specified by the user during initial system construction have joined the cluster.

During initial cluster construction, the state in which the cluster is waiting to be composed when all the nodes that make up the cluster have not been incorporated into the cluster is known as [INIT_WAIT]. When the number of nodes constituting a cluster has joined the cluster, the state will automatically change to the operating state.

There are 2 operating states. These are [STABLE] and [UNSTABLE].

- [STABLE] state
    - State in which a cluster has been formed by the number of nodes specified in the number of nodes constituting a cluster and service can be provided in a stable manner.
- [UNSTABLE] state
    - State in which the number of nodes constituting a cluster has not been fulfilled.
    - Cluster service will continue for as long as a majority of the number of nodes constituting a cluster is in operation.

A cluster can be operated in an [UNSTABLE] state as long as a majority of the nodes are in operation even if they are detached from a cluster due to maintenance and other reasons.

Cluster service is interrupted automatically in order to prevent a split brain from occurring when the number of nodes making up a cluster falls below the majority of the number of nodes constituting a cluster. The state in which cluster service has been interrupted is known as [WAIT] state.

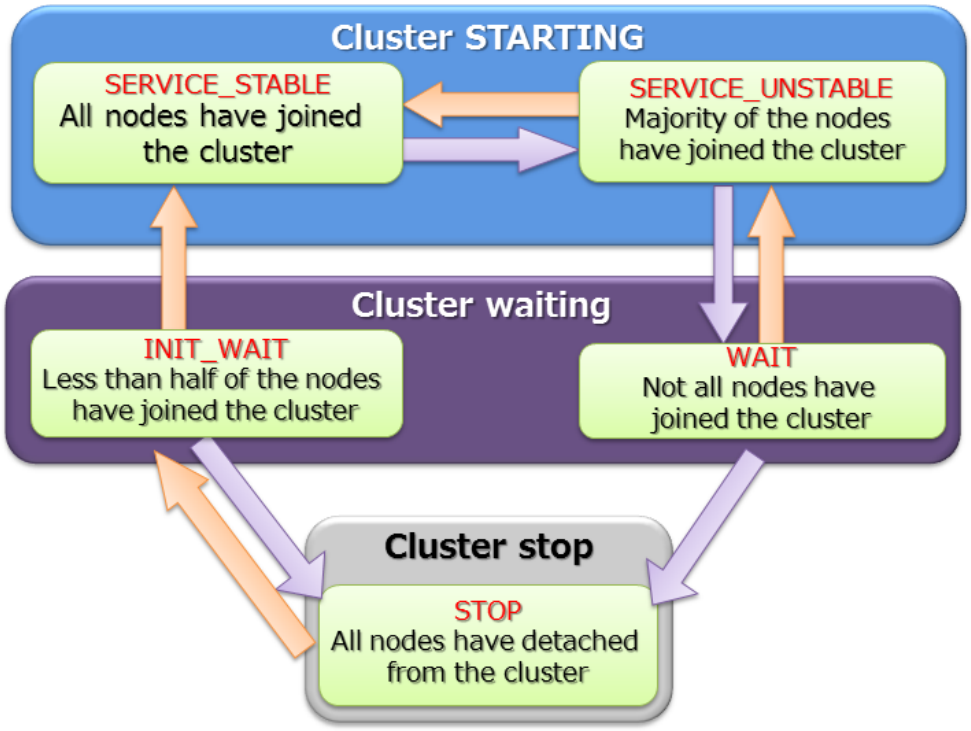
- A split brain

  is an action where multiple cluster systems performing the same process provide simultaneous service when a system is divided due to a hardware or network failure in a tightly-coupled system that works like a single server interconnecting multiple nodes. If the operation is continued in this state, data saved as replicas in multiple clusters will be treated as master data, resulting in data consistency being lost.

To restart cluster service from the [WAIT] state, new nodes are added to a cluster and nodes with errors are restored. The state will become [STABLE] once the number of nodes constituting a cluster has joined the cluster again.

When the number of clusters constituting a cluster falls below half due to a failure in a node constituting the cluster and the cluster operation is disrupted, new nodes are added to a cluster and nodes with errors are restored. Cluster service is automatically restarted once a majority of the nodes has joined the cluster.

However, if the operator intentionally uses the gs_leave command to separate the node from the cluster service e.g. during maintenance operations, and if the number of nodes constituting the cluster drops below half, cluster service will not be re-started even if additional nodes are added and more than half the nodes have joined the cluster. In this case, cluster service will not be re-started until the number of nodes constituting a cluster is reached.



Cluster status

A STABLE state is a state in which the value of the json parameter shown in gs_stat, /cluster/activeCount, is equal to the value of /cluster/designatedCount.

```
%gs_stat -u admin/admin -s
{
    "checkpoint": {
        "archiveLog": 0,
          :
          :
    },
    "cluster": {
        "activeCount":4,                    ★Nodes in operation within the cluster
        "clusterName": "test-cluster",
        "clusterStatus": "MASTER",
        "designatedCount": 4,               ★Number of nodes constituting a cluster
        "loadBalancer": "ACTIVE",
        "master": {
            "address": "192.168.0.1",
```

```
        "port": 10040
    },
    "nodeList": [                          ★Node list constituting a cluster
        {
            "address": "192.168.0.1",
            "port": 10040
        },
        {
            "address": "192.168.0.2",
            "port": 10040
        },
        {
            "address": "192.168.0.3",
            "port": 10040
        },
        {
            "address": "192.168.0.4",
            "port": 10040
        },

    ],
    :
    :
```

The status of the cluster can be checked with gs_sh or gs_admin. An example on checking the cluster status with gs_sh is shown below.

```
% gs_sh
gs> setuser admin admin gsadm                  // Setting connecting user
gs> setnode node1 192.168.0.1 10040            // Definition of a node constituting the cluster
gs> setnode node2 192.168.0.2 10040
gs> setnode node3 192.168.0.3 10040
gs> setnode node4 192.168.0.4 10040
```

```
gs> setcluster cluster1 test150 239.0.0.5 31999 $node1 $node2 $node3 $node4   //
Definition of cluster
gs> startnode $cluster1                    // Start-up of all nodes making up the cluster
gs> startcluster $cluster1                  // Instructing cluster composition
Waiting for cluster to start.
Cluster has started.
gs> configcluster  $cluster1                    ★Checking status of cluster
Name               : cluster1
ClusterName         : test-cluster
Designated Node Count : 4
Active Node Count    : 4
ClusterStatus        : SERVICE_STABLE           ★Stable state

Nodes:
  Name    Role Host:Port            Status
---------------------------------------------------
  node1     M  192.168.0.1:10040    SERVICING
  node2     F  192.168.0.2:10040    SERVICING
  node3     F  192.168.0.3:10040    SERVICING
  node4     F  192.168.0.4:10040    SERVICING

s> leavecluster $node2
Waiting for node to separate from cluster
Node has separated from cluster.
gs> configcluster  $cluster1
Name               : cluster1
ClusterName         : test150
Designated Node Count : 4
Active Node Count    : 3
ClusterStatus        : SERVICE_UNSTABLE         ★Unstable state

Nodes:
  Name    Role Host:Port            Status
---------------------------------------------------
  node1     M  192.168.0.1:10040    SERVICING       //Master node
  node2     -  192.168.0.2:10040    STARTED
```

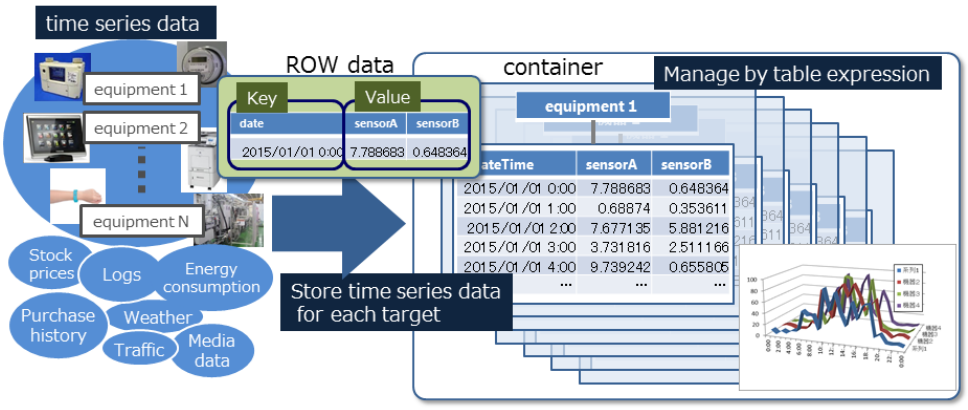
```
node3     F  192.168.0.3:10040    SERVICING       //Follower node
node4     F  192.168.0.4:10040    SERVICING       // Follower node
```

3.2 Data model

GridDB is a unique Key-Container data model that resembles Key-Value. It has the following features.

- A concept resembling a RDB table that is a container for grouping Key-Value has been introduced.
- A schema to define the data type for the container can be set. An index can be set in a column.
- Transactions can be carried out on a row basis within the container. In addition, ACID is guaranteed on a container basis.

GridDB manages data on a block, container, partition, and partition group basis.



Data model

GridDB manages data on a block, container, table, row, partition, and partition group basis.

- Block

  A block is a data unit for data persistence processing in a disk (hereinafter known as a checkpoint) and is the smallest physical data management unit in GridDB.
  Multiple container data are arranged in a block. Before initial startup of GridDB, a size of either 64 KB or 1 MB can be selected for the block size to be set up in the definition file

(cluster definition file). Specify 64 KB if the installed memory of the system is low, or if the frequency of data increase is low.

As a database file is created during initial startup of the system, the block size cannot be changed after initial startup of GridDB.

- Container

  A container consists of multiple blocks. A container is a data structure that serves as an interface with the user.

  There are 2 data types in a container, collection and time series.

- Table

  A table is a special container form that exists only in NewSQL products. SQL can be operated as an interface in NewSQL products.

  Before registering data in an application, there is a need to make sure that a container or table is created beforehand. Data is registered in a container or table.

- Row

  A row refers to a row of data to be registered in a container or table. Multiple rows can be registered in a container or table but this does not mean that data is arranged in the same block. Depending on the registration and update timing, data is arranged in suitable blocks within partitions.

  Normally, there are columns with multiple data types in a row.

- Partition

  A partition is a data management unit that includes 1 or more containers or tables.

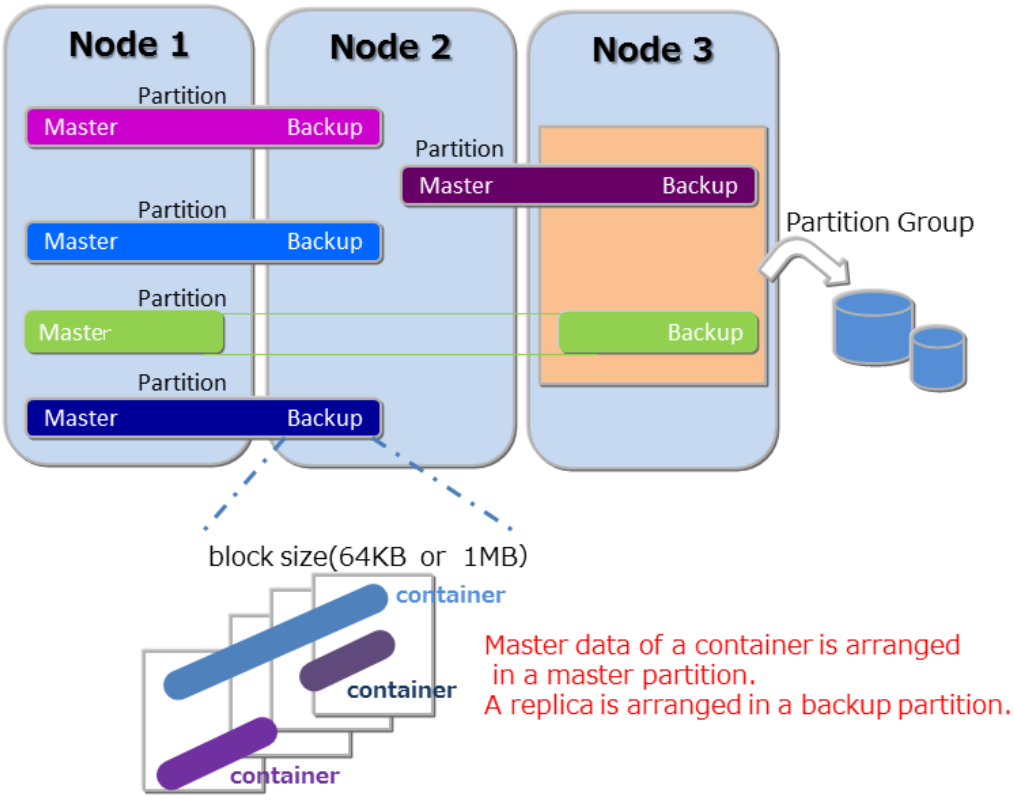
  A partition is a data arrangement unit between clusters for managing the data movement to adjust the load balance between nodes and data multiplexing (replica) in case of a failure. Data replica is arranged in a node to compose a cluster on a partition basis.

A node that can be updated against a container inside a partition is known as an owner node and 1 node is allocated to each partition. A node that maintains replicas other than owner nodes is a backup node. Master data and multiple backup data exist in a partition, depending on the number of replicas set.

- Partition group

  A group of multiple partitions is known as a partition group.

  Data maintained by a partition group is saved in an OS disk as a physical database file. A partition group is created with a number that depends on the degree of parallelism of the database processing threads executed by the node.



Data management unit

4. GridDB functions

Describes the data management functions, functions specific to the data model, operating functions and application development interfaces of GridDB.

4.1 Resource management

Besides the database in the memory, there are also resources constituting a GridDB cluster that are perpetuated in a disk. Perpetuated resources include the following.

- Database file

  A database file is a perpetuated file group to write data saved in a node constituting a cluster into a disk or SSD. A database file is a generic term to describe the transaction log file that is saved every time the GridDB database is updated and the checkpoint file that is written regularly by the database in the memory.

- Definition file

  There are 2 types of definition file, a parameter file (gs_cluster.json: hereinafter known as a cluster definition file) when composing a cluster, and a parameter file (gs_node.json: hereinafter known as a node definition file) to set the operations and resources of the node in the cluster. In addition, there is also a user definition file for GridDB administrator users.

- Event log file

  The operating log of the GridDB server is saved. Messages such as errors, warnings, etc. are saved.

- Backup file

  Backup data in the data file of GridDB is saved

The layout of these resources can be defined in GridDB home (path specified in environmental variable GS_HOME). In the initial installation state, the

/var/lib/GridStore directory is GridDB home, and the initial data of each resource is placed under this directory.

The initial configuration status is as follows.

```
/var/lib/GridStore/
    admin/
    backup/
    conf/
        gs_cluster.json
        gs_node.json
        password
    data/
    log/
```

The database directory, backup directory and server event log directory can be changed by changing the settings of the node definition file as well.

In a system that has multiple disk drives, be sure to change the definition information in order to prevent loss of backup data during a disk failure.

See Parameters for the contents that can be set in the cluster definition file and node definition file.

4.2 User management

There are 2 types of GridDB user, an OS user which is created during installation and a GridDB user to perform operations/development in GridDB (hereinafter known as a GridDB user).

4.2.1 OS user

An OS user has the right to execute operating functions in GridDB and a gsadm user is created during GridDB installation. This OS user is hereinafter known as gsadm.

All GridDB resources will become the property of gsadm. In addition, all operating commands in GridDB are executed by a gsadm.

A check is conducted to see whether the user has the right to connect to the GridDB server and execute the operating commands. This authentication is performed by a GridDB user.

4.2.2 GridDB user

- Administrator user and general user

  There are 2 types of GridDB user, an administrator user and a general user, which differ in terms of which functions can be used. Immediately after the installation of GridDB, 2 users, a system and an admin user, are registered as default administrator users.

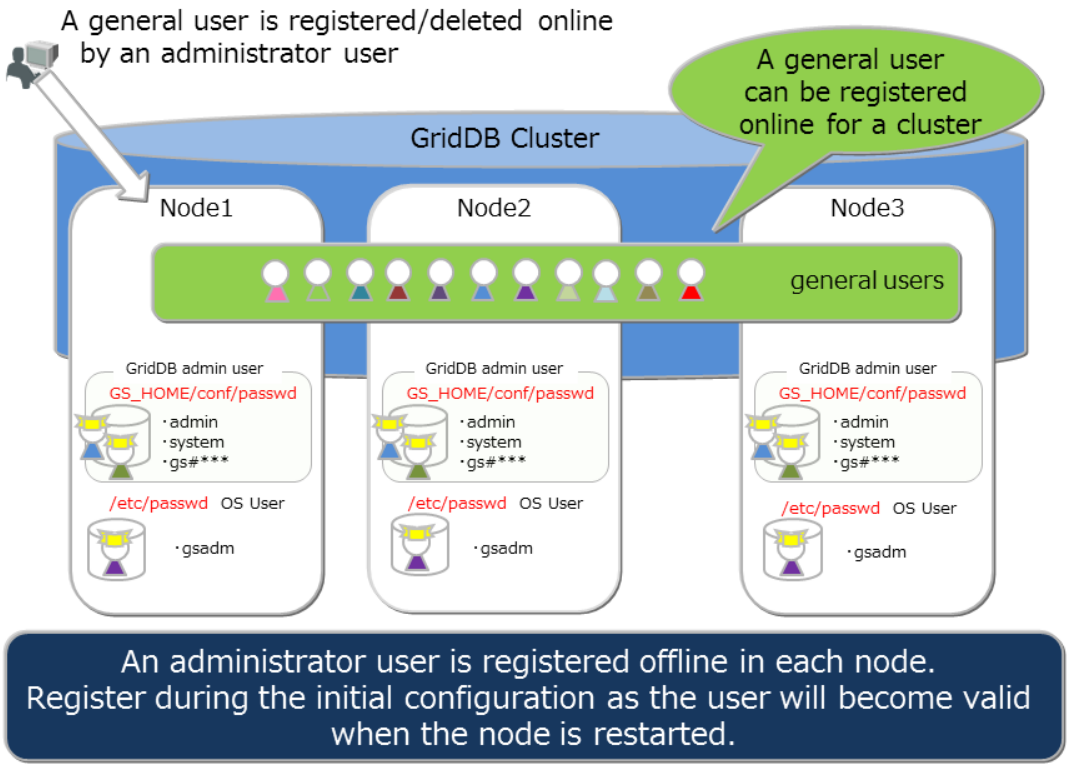
  An administrator user is a user created to perform GridDB operations while general users are users used by the application system.

  For security reasons, administrator users and general users need to be used differently according to the usage purpose.

- Creating a user

  An administrator user can register or delete a gsadm, and the information is saved in the password file of the definition file directory as a GridDB resource. As an administrator user is saved/managed in a local file of the OS, it has to be placed so that the settings are the same in all the nodes constituting the cluster. In addition, administrator users need to be set up prior to starting the GridDB server. After the GridDB server is started, administrative users are not valid even if they are registered.

  A general user can be created after an administrator user starts cluster operations in GridDB. A general user cannot be registered before the start of cluster services. A general user can only be registered using an operating command against a cluster as it is created after a cluster is composed in GridDB and maintained as management information in the GridDB database.



GridDB users

Since information is not communicated automatically among clusters, an administrator user needs to make the same settings in all the nodes and perform operational management such as determining the master management node of the definition file and distributing information from the master management node to all the nodes that constitute the cluster.

- Rules when creating a user

  There are naming rules to be adopted when creating a user name.

  - Administrator user: Specify a user starting with "gs#". After "gs#", the name should be composed of only alphanumeric characters and the underscore mark. Since the name is not case-sensitive, gs#manager and gs#MANAGER cannot be registered at the same time.
  - General user: Specify using alphanumeric characters and the underscore mark. However, the first character cannot be a number. In addition, since the name is not case-sensitive, user and USER cannot

be registered at the same time. System and admin users cannot be created as default administrator users.

- Password: No restrictions on the characters that can be specified.

A string consisting of up to 64 characters can be specified for the user name and password.

4.2.3 Usable function

The operations that can be carried out by an administrator and a general user are shown below. Among the operations, commands which can be executed by a gsadm without using a GridDB user are marked with "✓✓".

Operations	Operating details	Operating tools used	gsadm	Administrator user	General user
Node operations	Starting a node	gs_startnode /gs_sh		✓	✗
	Stopping a node	gs_stopnode /gs_sh		✓	✗
Cluster operations	Building a cluster	gs_joincluster /gs_sh		✓	✗
	Adding a note to a cluster	gs_addcluster /gs_sh		✓	✗
	Detaching a node from a cluster	gs_leavecluster /gs_sh		✓	✗
	Stopping a cluster	gs_stopcluster /gs_sh		✓	✗
User management	Registering an administrator	gs_adduser	✓✓	✗	✗

Operations	Operating details	Operating tools used	gsadm	Administrator user	General user
	user				
	Deleting an administrator user	gs_deluser	✓✓	✗	✗
	Changing the password of an administrator user	gs_passwd command	✓✓	✗	✗
	Creating a general user	gs_sh		✓	✗
	Deleting a general user	gs_sh		✓	✗
	Changing the password of a general user	gs_sh		✓	✓: Individual only
Database management	Creating/deleting a database	gs_sh		✓	✗
	Assigning/cancelling a user in the database	gs_sh		✓	✗
Data Operations	Creating/deleting a container or table	gs_sh		✓	✓: Only in the DB of the individual

Operations	Operating details	Operating tools used	gsadm	Administrator user	General user
	Registering data in a container or table	gs_sh		✓	✓:Only in the DB of the individual
	Searching for a container or table	gs_sh		✓	✓: Only in the DB of the individual
	Search operations in a container or table	gs_sh		✓	✓: Only in the DB of the individual
System status management	Acquiring system information	gs_stat		✓	✗

4.2.4 Database and user

Access to a cluster database (hereinafter known as cluster database) in GridDB can be separated on a user basis. The separation unit is known as a **database.** The following is a cluster database in the initial state.

- public
    - The database can be accessed by all administrator user and general users.
    - This database is used when connected without specifying the database at the connection point.

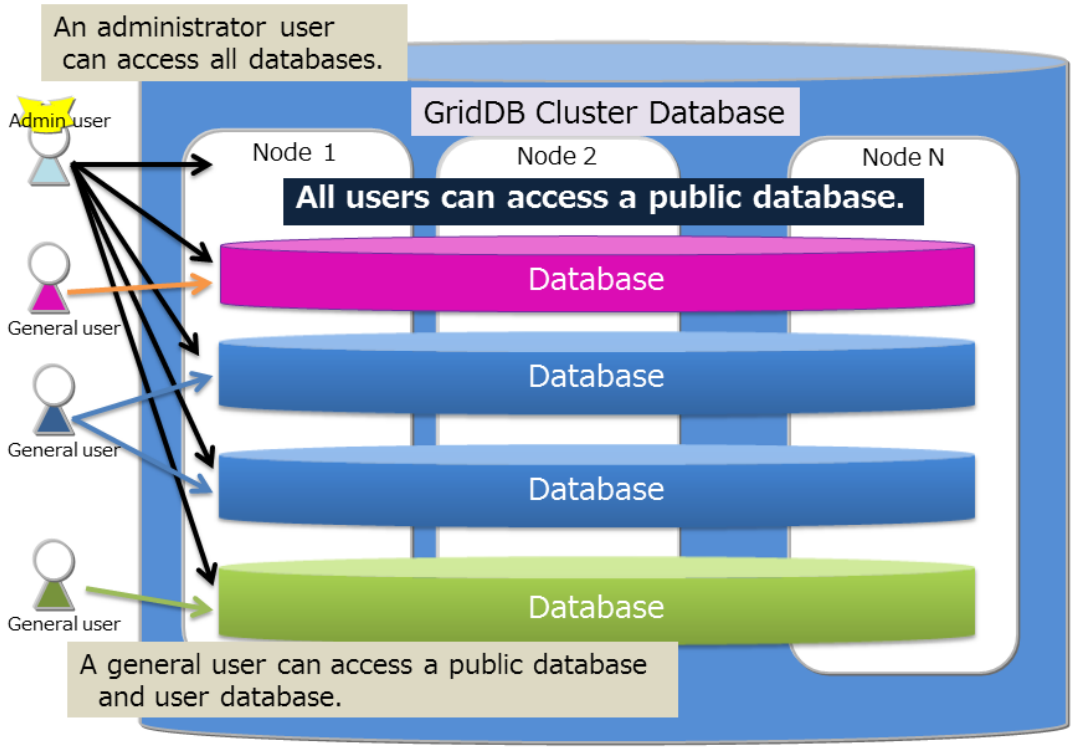
Multiple databases can be created in a cluster database. Creation of databases and assignment to users are carried out by an administrator user.

The rules for creating a database are as shown below.

- The maximum no. of users and the maximum no. of databases that can be created in a cluster database is 128.
- A string consisting of alphanumeric characters and the underscore mark can be specified for the database. However, the first character cannot be a number.
- A string consisting of 64 characters can be specified for the database name.
- Although the case sensitivity of the database name is maintained, a database which has the same name when it is not case-sensitive cannot be created.
- "public" and "information_schema" cannot be specified for default DB.

Only assigned general users and administrator users can access the database. Administrator user can access all databases. The following rules apply when assign a general user to a database.

- Only 1 general user can be assigned to 1 database
- Multiple databases can be assigned to 1 user



Database and users

The rules for creating a database are as shown below.

- The maximum no. of users and the maximum no. of databases that can be created in a cluster database is 128.
- Specify using alphanumeric characters and the underscore mark. However, the first character cannot be a number. In addition, since the name is not case-sensitive, database and DATABASE cannot be registered at the same time.

4.3 Data management function

To register and search for data in GridDB, a container or table (NewSQL products only) needs to be created to store the data. This section describes the data types that can be registered in a container or table, data size, index and data management functions.

The naming rules for containers and tables are the same as those for databases..

- A string consisting of alphanumeric characters and the underscore mark can be specified. However, the first character cannot be a number.
- Although the case sensitivity of the name is maintained, a container (table) which has the same name when it is not case-sensitive cannot be created.

4.3.1 Container data type

There are 2 container data types.
A **timeseries container** is a data type which is suitable for managing hourly data together with the occurrence time while a **collection** is suitable for managing a variety of data.

The schema can be set in a container.

The basic data types that can be registered in a container are the **basic data type** and **array data type.**

- Basic data types
  Describes the basic data types that can be registered in a container. A basic data type cannot be expressed by a combination of other data types.

Data type	Description
BOOLEAN	True or false
STRING	Composed of an arbitrary number of characters using the unicode code point
BYTE	Integer value from -2^{7} to 2^{7}-1 (8 bits)
SHORT	Integer value from -2^{15} to 2^{15}-1 (16 bits)
INTEGER	Integer value from -2^{31} to 2^{31}-1 (32 bits)
LONG	Integer value from -2^{63} to ら2^{63}-1 (64 bits)
FLOAT	Single-precision data type (32 bits) Floating-point number defined in IEEE754
DOUBLE	Double-precision data type (64 bits) Floating-point number defined in IEEE754
TIMESTAMP	Data type expressing the date and time Data format maintained in the database is UTC, and accuracy is in milliseconds
GEOMETRY	Data type to represent a space structure
BLOB	Data type for binary data such as images, audio, etc.

- The following restrictions apply to the size of the data that can be managed for STRING, GEOMETRY and BLOB data. The restriction value varies according to the block size which is the input/output unit of the database in the GridDB definition file (gs_node.json).

Data type	Block size (64KB)	Block size (1MB)
STRING	Maximum 31KB (equivalent to UTF-8 encode)	Maximum 128KB (equivalent to UTF-8 encode)
GEOMETRY	Maximum 31KB (equivalent to	Maximum 128KB (equivalent to

Data type	Block size (64KB)	Block size (1MB)
	the internal storage format)	the internal storage format)
BLOB	Maximum 127MB	Maximum 1GB

- HYBRID
  A data type composed of a combination of basic data types that can be registered in a container. The only hybrid data type in the current version is an array.
    - ARRAY

  Expresses an array of values. Among the basic data types, only GEOMETRY and BLOB data cannot be maintained as an array. The restriction on the data volume that can be maintained in an array varies according to the block size of the database.

Data type	Block size (64KB)	Block size (1MB)
Number of arrays	4000	65000

[Memo]

The following restrictions apply to TQL operations in an array column.

- Although the i-th value in the array column can be compared, calculations (aggregation) cannot be performed on all the elements.

*(Example) When column A is an array and assumed to be defined

- The elements in an array such as select * where ELEMENT (0, column A) > 0 can be specified and compared. However, the variable in the ELEMNT "0" section cannot be specified.
- Aggregation such as select SUM (column A) cannot be carried out.

4.3.2 Container ROWKEY

A ROWKEY is the data set in the row of a container. The uniqueness of a row with a set ROWKEY is guaranteed.

A ROWKEY can be set in the first column of the row. (This is set in Column No. 0 since columns start from 0 in GridDB.)

- For a timeseries container
    - ROWKEY is a TIMESTAMP
    - Must be specified.
- For a collection
    - A ROWKEY is either a STRING, INTEGER, LONG or TIMESTAMP column.
    - Need not be specified.

A default index prescribed in advance according to the column data type can be set in a column set in ROWKEY.

In the current version, the default index of all STRING, INTEGER, LONG or TIMESTAMP data that can be specified in a ROWKEY is the TREE index.

4.3.3 Container index

A condition-based search can be processed quickly by creating an index for the columns of a container.

There are 3 types of index - hash index (HASH), tree index (TREE) and space index (SPATIAL). A hash index is used in an equivalent-value search when searching with a query in a container. Besides equivalent-value search, a tree index is used in comparisons including the range (bigger/same, smaller/same etc.).

The index that can be set differs depending on the container type and column data type.

- HASH INDEX
    - An equivalent value search can be conducted quickly but this is not suitable for searches that read the rows sequentially.

    - Columns of the following data type can be set in a collection. Cannot be set in a timeseries container.
        - STRING
        - BOOL
        - BYTE
        - SHORT
        - INTEGER
        - LONG
        - FLOAT
        - DOUBLE
        - TIMESTAMP
- Besides equivalent-value search, a tree index
    - is used in comparisons including the range (bigger/same, smaller/same etc.).
    - This can be used for columns of the following data type in any type of container, except for columns corresponding to a rowkey in a timeseries container.
        - STRING
        - BOOL
        - BYTE
        - SHORT
        - INTEGER
        - LONG
        - FLOAT
        - DOUBLE
        - TIMESTAMP
- SPACE INDEX
    - Can be used for only GEOMETRY columns in a collection. This is specified when conducting a spatial search at a high speed.

Although there are no restrictions on the no. of indices that can be created in a container, creation of an index needs to be carefully designed. An index is updated when the rows of a configured container are inserted, updated or deleted. Therefore, when multiple indices are created in a column of a row that is updated frequently, this will affect the performance in insertion, update or deletion operations.

An index is created in a column as shown below.

- A column that is frequently searched and sorted.
- A column that is frequently used in the condition of the WHERE section of TQL
- High cardinality column (containing few duplicated values)

4.3.4 Timeseries container

In order to manage data from a sensor etc. occurring at a high frequency, data is placed in accordance with the data placement algorithm (TDPA: Time Series Data Placement Algorithm) making maximum effective use of the memory . In a timeseries container, memory is allocated while classifying internal data by its periodicity. When hint information is given in an affinity function, the placement efficiency rises further. Expired data in a timeseries container is released at almost zero cost while being expelled to a disk where necessary.

A timeseries container has a TIMESTAMP ROWKEY.

- Expiry release function
  In a timeseries data, an expiry release function and the data retention period can be set so that when the period set is exceeded, the data will be released (deleted).

  The settings refer to the deadline unit and deadline, and no. of divisions when the data is released during container creation. The settings of a timeseries container that has been created cannot be changed.

  The deadline can be set in day/hour/minute/sec/millisec units. The year unit and month unit cannot be specified. The current time used in determining whether the valid period has expired is dependent on the execution environment of each node in GridDB. Therefore, if the GridDB node time is faster than the client time due to a network delay or a deviation in the time setting of the execution environment, or if a row prior to expiry is no longer accessible, or conversely if only the client time is faster, the expired row may be accessible.
  We recommend that a value larger than the minimum required time is set in order to avoid unintended loss of rows.

  An expired row is deemed as non-existent and is no longer subject to row operations such as search and update.

Expired rows are physically deleted based on the number of divisions for the valid period (number of divisions when the data is deleted).

For example, if the valid period is 720 days and the specified number of divisions is 36, although data access will be immediately disabled upon passing the 720-day mark, the data will only be deleted after 20 days have passed from the 720 days. 20 days' worth of physical data is deleted together.

The number of divisions is specified when creating a container.

- Calculation of a timeseries container
  There are calculations to perform time correction in addition to calculations to aggregate containers in a timeseries container.
  - In an aggregate operation on an aggregate operation container, specify the start and end time and perform the aggregate operation on a row set or specific column.
  - Aggregate operation specific to a timeseries container

    In a timeseries container, the calculation is performed with the data weighted at the time interval of the sampled data. In other words, if the time interval is long, the calculation is carried out assuming the value is continued for an extended time.

4.3.5 Selection and interpolation of a timeseries container

Time data may deviate slightly from the expected time due to the timing of the collection and the contents of the data to be collected. Therefore when conducting a search using time data as a key, a function that allows data around the specified time to be acquired in also required.

4.3.6 Affinity function

An affinity is a function to connect related data. There are 2 types of affinity function in GridDB, data affinity and node affinity.

- Data affinity function
  A data affinity is a function to raise the memory hit rate by arranging highly correlated data in the same block and localizing data access. By raising the

memory hit ratio, the no. of memory mishits during data access can be reduced and the throughput can be improved. By using data affinity, even machines with a small memory can be operated effectively.

The data affinity settings provide hint information as container properties when creating a container. The characters that can be specified for the hint information are restricted by naming rules that are similar to those for the container name. Data with the same hint information is placed in the same block as much as possible.

Data affinity hints are set separately by the data update frequency and reference frequency. For example, consider the data structure when system data is registered, referenced or updated by the following operating method in a system that samples and refers to the data on a daily, monthly or annual basis in a monitoring system.

1. Data in minutes is sent from the monitoring device and saved in the container created on a monitoring device basis.
2. Since data reports are created daily, one day's worth of data is aggregated from the data in minutes and saved in the daily container
3. Since data reports are created monthly, daily container data is aggregated and saved in the monthly container
4. Since data reports are created annually, monthly container data is aggregated and saved in the annual container
5. The current space used (in minutes and days) is constantly updated and displayed in the display panel.

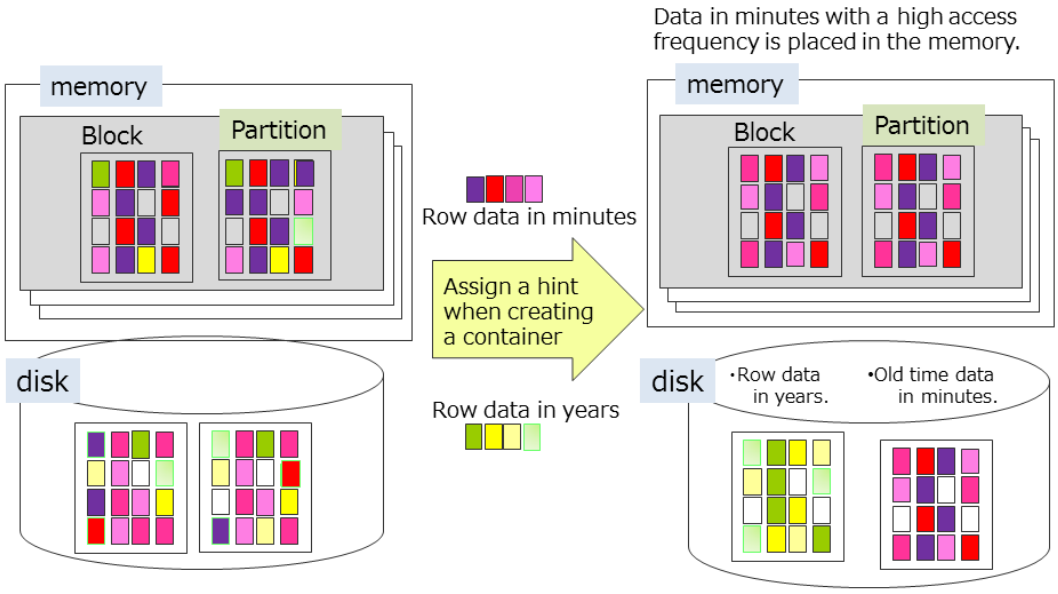
In GridDB, instead of occupying a block in a container unit, data close to the time is placed in the block. Therefore, refer to the daily container in 2., perform monthly aggregation and use the aggregation time as a ROWKEY. The data in 3. and the data in minutes in 1. may be saved in the same block.

If the memory is small and the data is so big that all the monitoring data cannot be stored in the memory, when the aggregation process in 4. is carried out on an annual basis, the block is divided and data placed in 3. is placed in the memory. As a result, data that you want to monitor may get swapped out

as the data read may not be the latest e.g. data in 1. which is not required all the time is driven out of the memory.

In this case, by providing hints to the container according to the container access frequency using a data affinity e.g. on a minute, daily or monthly basis, etc., data with a low access frequency and data with a high access frequency is separated into different blocks when the data is placed.

In this way, data can be placed to suit the usage scene of the application by the data affinity function.

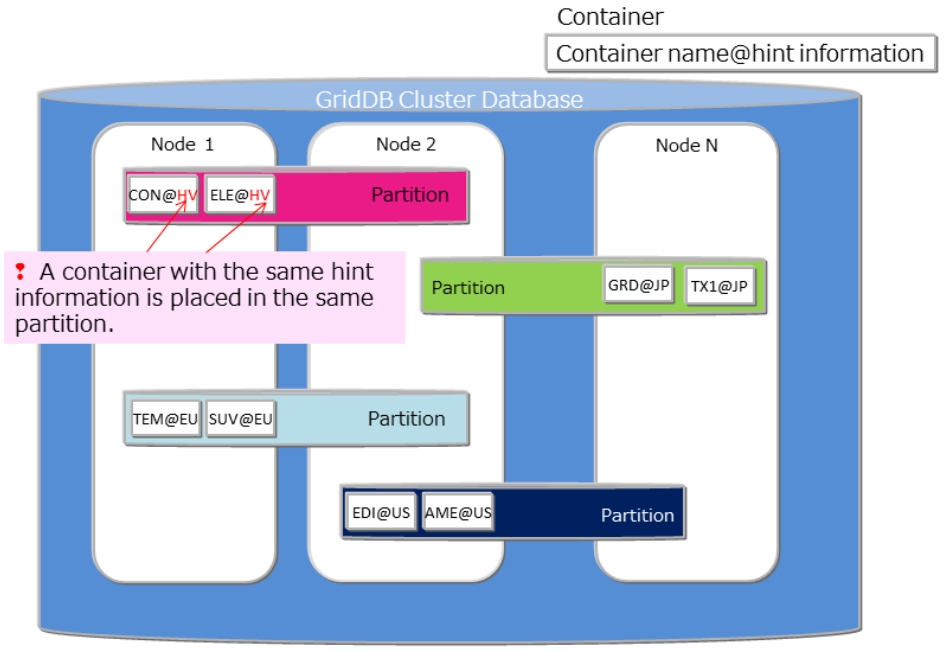


Data with a high data occurrence frequency or reference frequency due to data affinity is placed in the same block

Data Affinity

- Node affinity function

  Node affinity is a function to reduce the network load when accessing data by arranging highly correlated containers and tables in the same node. Although there is no container JOIN operation In the TQL of a NoSQL product, a table JOIN operation can be described in the SQL of a SQL product. When joining a table, the network access load of a table placed in another node of the cluster can be reduced. In addition, since concurrent processing using multiple nodes is no longer possible, there is no effect on shortening the turnaround time. Nonetheless, throughput may still rise due to a reduction in the network load.



Placement of container/table based on node affinity

To use the node affinity function, hint information is given in the container name when the container is created. A container with the same hint information is placed in the same partition. Specify the container name as shown below.

- Container name@node affinity hint information

The naming rules for node affinity hint information are the same as the naming rules for the container name.

4.4 Transaction processing

GridDB supports transaction processing on a container basis and ACID characteristics which are generally known as transaction characteristics. The supporting functions in a transaction process are explained in detail below.

4.4.1 Starting and ending a transaction

When a row search or update etc. is carried out on a container, a new transaction is started and this transaction ends when the update results of the data are committed or aborted.

[Memo]

- A commit is a process to confirm transaction information under processing to perpetuate the data.
    - In GridDB, updated data of a transaction is stored as a transaction log by a commit process, and the lock that had been maintained will be released.
- An abort is a process to rollback (delete) all transaction data under processing.
    - In GridDB, all data under processing are discarded and retained locks will also be released.

The initial action of a transaction is set in autocommit.

In autocommit, a new transaction is started every time a container is updated (data addition, deletion or revision) by the application, and this is automatically committed at the end of the operation. A transaction can be committed or aborted at the requested timing by the application by turning off autocommit.

A transaction recycle may terminate in an error due to a timeout in addition to being completed through a commit or abort. If a transaction terminates in an error due to a timeout, the transaction is aborted. The transaction timeout is the elapsed time from the start of the transaction. Although the initial value of the transaction timeout time is set in the definition file (gs_node.json), it can also be specified as a parameter when connecting to GridDB on an application basis.

4.4.2 Transaction consistency level

There are 2 types of transaction consistency levels, immediate consistency and eventual consistency. This can also be specified as a parameter when connecting to GridDB for each application. The default setting is immediate consistency.

- Immediate consistency: Container update results from other clients are reflected immediately at the end of the transaction concerned. As a result, the latest details can be referenced all the time.
- Eventual consistency: Container update results from other clients may not be reflected immediately at the end of the transaction concerned. As a result, there is a possibility that old details may be referred to.

Immediate consistency is valid in update operations and read operations. Eventual consistency is valid in read operations only. For applications which do not require the latest results to be read all the time, the reading performance improves when eventual consistency is specified.

4.4.3 Transaction isolation level

Conformity of the database contents need to be maintained all the time. When executing multiple transaction simultaneously, the following events will generally surface as issues.

- An event which involves uncommitted data written by a dirty read transaction being read by another transaction.
- An event which involves data read previously by a non-recurrent read transaction becoming unreadable.

Even if you try to read the data read previously by a transaction again, the previous data can no longer be read as the data has already been updated and committed by another transaction (the new data after the update will be read instead).

- An event in which the inquiry results obtained previously by a phantom read transaction can no longer be acquired.

Even if you try to execute an inquiry executed previously in a transaction again in the same condition, the previous results can no longer be acquired as the data satisfying the inquiry condition has already been changed, added and committed by another transaction (new data after the update will be acquired instead).

In GridDB, “READ_COMMITTED” is supported as a transaction isolation level. In READ_COMMITTED, the latest data confirmed data will always be read.

When executing a transaction, this needs to be taken into consideration so that the results are not affected by other transactions. The isolation level is an indicator from 1 to 4 that shows how isolated the executed transaction is from other transactions (the extent that consistency can be maintained).

The 4 isolation levels and the corresponding possibility of an event raised as an issue occurring during simultaneous execution are as follows.

Isolation level	Dirty read	Non-recurrent reading	Phantom read
READ_UNCOMMITTED	Possibility of occurrence	Possibility of occurrence	Possibility of occurrence
READ_COMMITTED	Safe	Possibility of occurrence	Possibility of occurrence
REPEATABLE_READ	Safe	Safe	Possibility of occurrence
SERIALIZABLE	Safe	Safe	Safe

In READ_COMMITED, if data read previously is read again, data that is different from the previous data may be acquired, and if an inquiry is executed again, different results may be acquired even if you execute the inquiry with the same search condition. This is because the data has already been updated and committed by another transaction after the previous read.

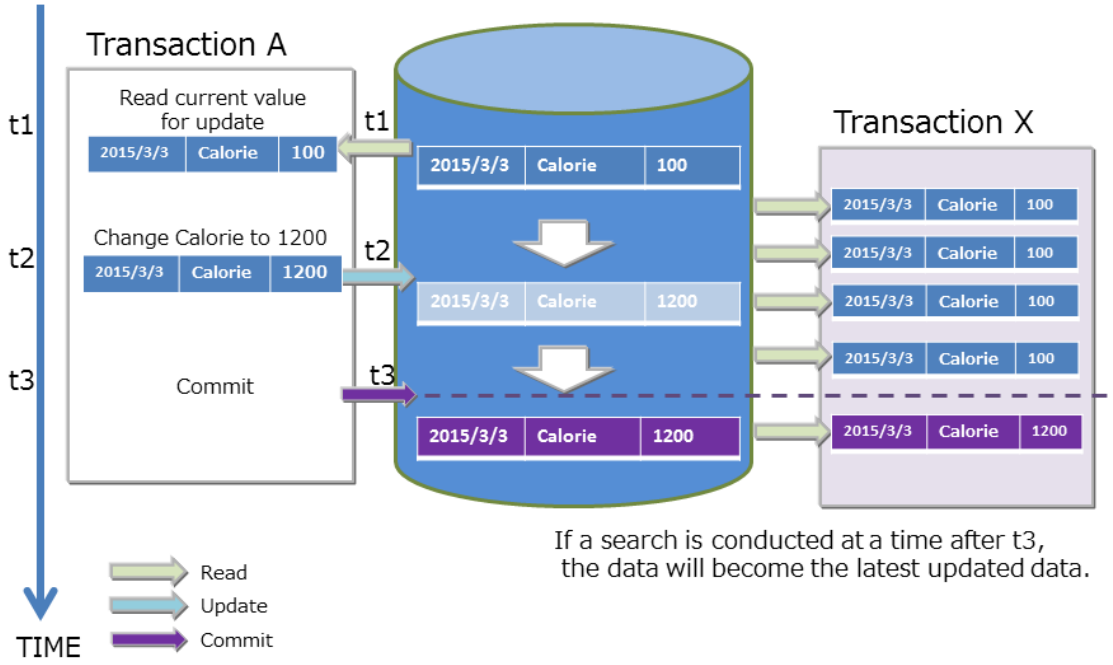
In GridDB, data that is being updated by MVCC is isolated.

4.4.4 MVCC

In order to realize READ_COMMITTED, “MVCC (Multi-Version Concurrency Control)” has been adopted.

MVCC is a processing method that refers to the data prior to being updated instead of the latest data that is being updated by another transaction when a transaction sends an inquiry to the database. System throughput improves as the transaction can be executed concurrently by referring to the data prior to the update.

When the transaction process under execution is committed, other transactions can also refer to the latest data.



MVCC

4.4.5 Lock

There is a data lock mechanism to maintain the consistency when there are competing container update requests from multiple transactions.

The lock granularity differs depending on the type of container. In addition, the lock range changes depending on the type of operation in the database.

- Lock granularity
  - A timeseries container is a data structure to hold data that is being generated with each passing moment and rarely includes cases in which the data is updated at a specific time.
  - Collection data may include cases in which an existing ROW data is updated as it manages data just like a RDB table.

  Based on the use case analysis of such a container, the lock granularity (smallest unit) adopted in GridDB is as follows. The lock granularity of a collection which is updated relatively more frequently is a ROW in order to improve the concurrent execution performance.

  - Collection･･･Lock by ROW unit.

- Timeseries container･･･Locked by ROW collection
    - In a row set, multiple rows are placed in a timeseries container by dividing a block into several data processing units. This data processing unit is known as a row set. It is a data management unit to process a large volume of timeseries containers at a high speed even though the data granularity is coarser than the lock granularity in a collection.

  The lock granularity of a collection which is updated randomly more frequently compared to a timeseries container collection adopts a row unit in order to improve the concurrent execution performance.

- Lock range by database operations
  Container operations are not limited to just data registration and deletion but also include schema changes accompanying a change in data structure, index creation to improve speed of access, and other operations. The range of the lock differs between an operation on a specific row of the container and an operation on all rows of the container.
    - Lock equivalent of a container unit
        - Index operations (createIndex/dropIndex)
        - Container deletion
        - Schema change
    - Lock in accordance with the lock granularity
        - insert/update/remove
        - get(forUpdate)

      In a data operation on a row, a lock following the lock granularity is ensured.

- If there is competition in securing the lock, the subsequent transaction will be put in standby for securing the lock until the earlier transaction has been completed by a commit or rollback process and the lock is released.
- A standby for securing a lock can also be cancelled by a timeout besides completing the execution of the transaction.

4.4.6 Data perpetuation

Data registered or updated in a container or table is perpetuated in the disk or SSD, and protected from data loss when a node failure occurs. There are 2 types of transaction log process, one to synchronize data in a data update and write the updated data sequentially in a transaction log file, and the other is a checkpoint process to store updated data in the memory regularly in the database file on a block basis.

To write to a transaction log, either one of the following settings can be made in the node definition file.

- 0: SYNC
- An integer value of 1 or higher1: DELAYED_SYNC

In the "SYNC" mode, log writing is carried out synchronously every time an update transaction is committed or aborted. In the "DELAYED_SYNC" mode, log writing during an update is carried out at a specified delay of several seconds regardless of the update timing. Default value is "1 (DELAYED_SYNC 1 sec)".

When "SYNC" is specified, although the possibility of losing the latest update details when a node failure occurs is lower, the performance is affected in systems that are updated frequently.

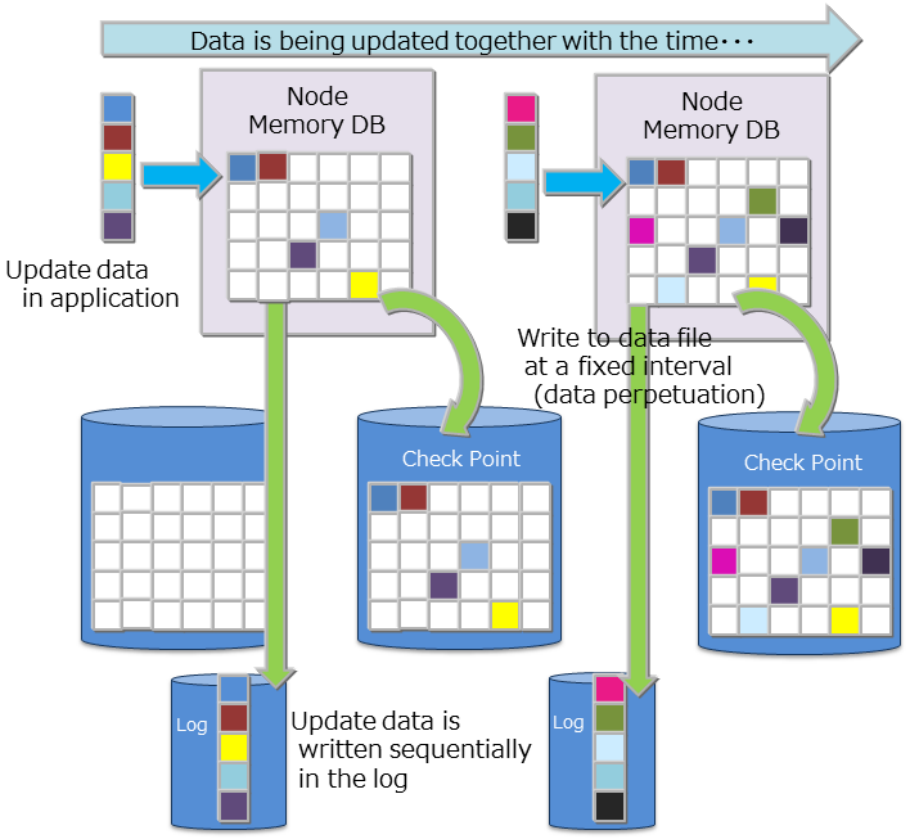
On the other hand, if "DELAYED_SYNC" is specified, although the update performance improves, any update details that have not been written in the disk when a node failure occurs will be lost.

If there are 2 or more replicas in a raster configuration, the possibility of losing the latest update details when a node failure occurs is lower even if the mode is set to "DELAYED_SYNC" as the other nodes contain replicas. Consider setting the mode to "DELAYED_SYNC" as well if the update frequency is high and performance is required.

In a checkpoint, the update block is updated in the database file. A checkpoint process operates at the cycle set on a node basis. A checkpoint cycle is set by the parameters in the node definition file. Initial value is 1200 sec (20 minutes).

By raising the checkpoint execution cycle figure, data perpetuation can be set to be carried out in a time band when there is relatively more time to do so e.g. by perpetuating data to a disk at night and so on. On the other hand, when the cycle is lengthened, the disadvantage is that the number of transaction log files that have to be rolled forward when a node is restarted outside the system process increases, thereby increasing the recovery time.

Data that is updated in a checkpoint execution is pooled and maintained in a memory separate from the checkpoint writing block. Set checkpoint concurrent execution for the checkpoint to carry out the checkpoint quickly. If concurrent execution is set, concurrent processing is carried out until the number of transactions executed simultaneously is the same.



CheckPoint

4.4.7 Timeout process

The timeout details that can be set differ between a NoSQL I/F and a NewSQL I/F.

- NoSQL timeout
  There are 2 types of timeout in a NoSQL that the application developer is kept informed of. There are 2 types of timeout, a transaction timeout that is related to the processing time limit of a transaction and a failover timeout that is related to the retry time of a recovery process when a failure occurs.
  - TransactionTimeout

    The timer is started when access to the container subject to the process begins, and a timeout occurs when the specified time is exceeded.

    Timeout time prepared to delete the lock and memory from a transaction possessing an extended update lock (application searches for data in the update mode and does not delete the data when the lock is maintained) or a transaction maintaining a large amount of results for an extended time (application does not delete the memory of the cluster system for an extended time) and so on. Application is aborted upon reaching the transaction timeout.

    Besides the node definition file, a transaction timeout can also be specified in the application with a parameter during cluster connection. The specification in the application is prioritized. The default transaction timeout setting is 0 sec. 0 sec means that there is no timeout specified. In order to monitor an extended transaction, set the timeout time to meet the system requirements.

  - FailoverTimeout

    Timeout time during an error retry when a client connected to a node constituting a cluster which failed connects to a replacement node. If a new connection point is discovered in the retry process, the client application will not be notified of the error. Default value is 5 minutes. This can also be specified in the application by a parameter during cluster connection. Failover timeout is also used in timeout during initial connection.

- Both the transaction timeout and failover timeout can be set when connecting to a cluster using a GridDB object in the Java API or C API. See "GridDB API Reference" (GridDB_API_Reference.html) for details.

4.4.8 Replication function

Data replicas are created on a partition basis in accordance with the number of replications set by the user among multiple nodes constituting a cluster.

A process can be continued non-stop even when a node failure occurs by maintaining replicas of the data among scattered nodes. In the client API, when a node failure is detected, the client automatically switches access to another node where the replica is maintained.

The default number of replication is 2, allowing data to be replicated twice when operating in a cluster configuration with multiple nodes.

When there is an update in a container, the owner node (the node having the master replica) among the replicated partitions is updated.

There are 2 ways of subsequently reflecting the updated details from the owner node in the backup node.

- Replication is carried out without synchronizing with the timing of the non-synchronous replication update process. Update performance is better for quasi-synchronous replication but the availability is worse.
- Although replication is carried out synchronously at the quasi-synchronous replication update process timing, no appointment is made at the end of the replication. Availability is excellent but performance is inferior.

If performance is more important than availability, set the mode to non-synchronous replication and if availability is more important, set it to quasi-synchronous replication.

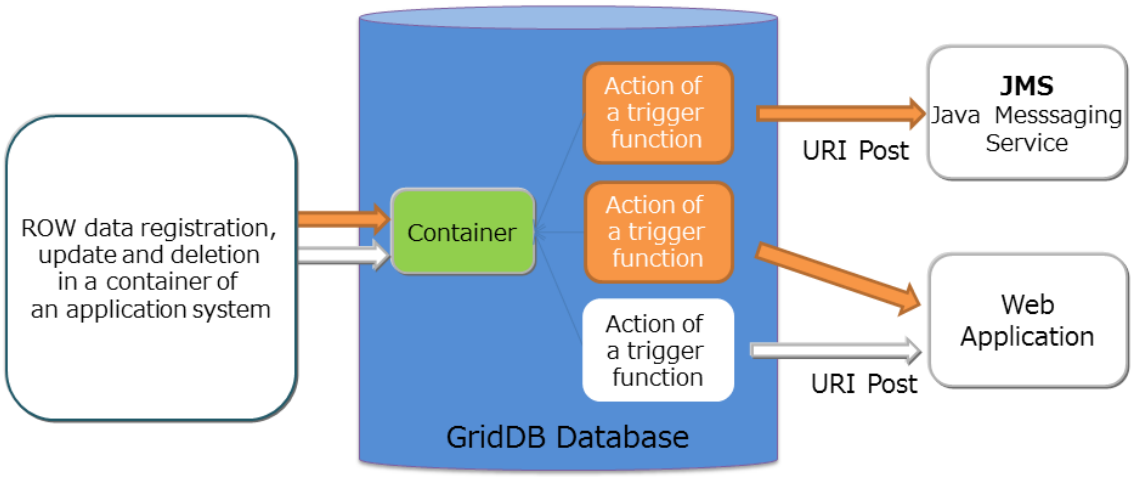
[Memo] The number of replications is set in the cluster definition file (gs_cluster.json) /cluster/replicationNum. Synchronous settings of the replication are set in the cluster definition file (gs_cluster.json) /transaction/replicationMode.

4.5 Trigger function

A trigger function is an automatic notification function when an operation (add/update or delete) is carried out on the row data of a container. Event notifications can be received without the need to poll and monitor database updates in the application system.

There are 2 ways of notifying the application system.

- Java Messaging Service(JMS)
- REST



Action of a trigger function

When a trigger occurs, the application can also be notified of the column data in a row data subject to the operation. As to which column data to notify, this is set when the trigger is set in the container. In addition, multiple triggers can also be set in a single container.

The items that can be specified with a trigger setting are as follows.

- Notification event condition (add/update or delete)
- Notification method (JMS or REST)
- Notification column

4.6 Failure process function

In GridDB, recovery for a single point failure is not necessary as replicas of the data are maintained in each node constituting the cluster. The following action is carried out when a failure occurs in GridDB.

1. When a failure occurs, the failure node is automatically isolated from the cluster.
2. Failover is carried out in the backup node in place of the isolated failure node.
3. Partitions are rearranged autonomously as the number of nodes decreases as a result of the failure (replicas are also arranged).

A node that has been recovered from a failure can be incorporated online into a cluster operation. A node can be incorporated into a cluster which has become unstable due to a failure using the gs_joincluster command. As a result of the node incorporation, the partitions will be rearranged autonomously and the node data and load balance will be adjusted.

In this way, although advance recovery preparations are not necessary in a single failure, recovery operations are necessary when operating in a single configuration or when there are multiple overlapping failures in the cluster configuration.

When operating in a cloud environment, even when physical disk failure or processor failure is not intended, there may be multiple failures such as a failure in multiple nodes constituting a cluster, or a database failure in multiple nodes.

4.6.1 Type and treatment of failures

An overview of the failures which occur and the treatment method is shown in the table below.

A node failure refers to a situation in which a node has stopped due to a processor failure or an error in a GridDB server process, while a database failure refers to a situation in which an error has occurred in accessing a database placed in a disk.

Configuration of GridDB	Type of failure	Action and treatment
Single configuration	Node failure	Although access from the application is no longer possible, data in a transaction which has completed processing can be recovered simply by restarting the transaction, except when caused by a node failure. Recovery by another node is considered when the node failure is prolonged.
Single configuration	Database failure	The database file is recovered from the backup data in order to detect an error in the application. Recovered at the backup point.
Cluster configuration	Single node failure	The error is covered up in the application, and the process can continue in nodes with replicas. Recovery operation is not necessary in a node where a failure has occurred.
Cluster configuration	Multiple node failure	If both owner/backup partitions of a replica exist in a failure target node , the cluster will operate normally even though the subject partitions cannot be accessed. Except when caused by a node failure, data in a transaction which has completed processing can be recovered simply by restarting the transaction. Recovery by another node is considered when the node failure is prolonged.
Cluster configuration	Single database failure	Since data access will continue through another node constituting the cluster when there is a database failure in a single node, the data can be recovered simply by changing the database deployment location to a different disk, and then starting the node again.
Cluster configuration	Multiple database failure	A partition that cannot be recovered in a replica needs to be recovered at the point backup data is sampled from the latest backup data.

4.6.2 Client failover

If a node failure occurs when operating in a cluster configuration, the partitions (containers) placed in the failure node cannot be accessed. At this point, a client failover function to automatically connect to the backup node again and continue the process is activated in the client API. To automatically perform a failover countermeasure in the client API, the application developer does not need to be aware of the error process in the node.

However, due to a network failure or simultaneous failure of multiple nodes, an error may also occur and access to the target application operations may not be possible.

Depending on the data to be accessed, the following points need to be considered in the recovery process after an error occurs.

- For a collection in which the timeseries container or row key is defined, the data can be recovered by executing the failed operation or transaction again.
- For a collection in which the row key is not defined, the failed operation or transaction needs to be executed again after checking the contents of the DB.

[Memo]

In order to simplify the error process in an application, it is recommended that the row key be defined when using a collection. If the data cannot be uniquely identified by a single column value but can be uniquely identified by multiple column values, a column having a value that links the values of the multiple columns is recommended to be set as the row key so that the data can be uniquely identified.

4.6.3 Event log function

An event log is a log to record system operating information and messages related to event information e.g. exceptions which occurred internally in a GridDB node etc.

An event log is created with the file name GridStore-%Y%m%d-n.log in the directory shown in the environmental variable eGS_LOG (Example: GridStore-20150328-5.log). The file is switched when the node is restarted or when the log output size exceeds a fixed size.

Output format of event log is as follows.

- (Date and time) (host name) (thread no.) (log level) (category) [(error trace no.): (error trace no. and name)] (message) < (base64 detailed information: Detailed information for problem analysis in the support service)>

  An overview of the event which occurred can be found in the error trace no. and name. In addition, measures to deal with the problems can be searched using the error trace no. in the troubleshooting guide. A output example of an event log is shown below.

```
2014-11-12T10:35:29.746+0900 TSOL1234 8456 ERROR TRANSACTION_SERVICE
[10008:TXN_CLUSTER_NOT_SERVICING] (nd={clientId=2, address=127.0.0.1:52719},
pId=0, eventType=CONNECT, stmtId=1)
<Z3JpZF9zdG9yZS9zZXJ2ZXIvdHJhbnNhY3Rpb25fc2VydmljZS5jcHAgQ29ubmVjdEhhbmRsZX
I6OmhhbmRsZUVycm9yIGxpbmU9MTg2MSA6IGJ5IERlbnlFeGNlcHRpb24gZ3JpZF9zdG9yZS9z
ZXJ2ZXIvdHJhbnNhY3Rpb25fc2VydmljZS5jcHAgU3RhdGVtZW50SGFuZGxlcjo6Y2hlY2tFeGVjd
XRhYmxlIGxpbmU9NjExIGNvZGU9MTAwMDg=>
```

4.7 Data access

To access GridDB data, there is a need to develop an application using JDBC or ODBC for NoSQL products' client API (Java, C language) or NewSQL products. Data can be accessed simply by connecting to the cluster database of GridDB without having to take into account position information on where the container or table is located in the cluster database. The application system does not need to consider which node constituting the cluster the container is placed in.

In the GridDB API, when connecting to a cluster database initially, placement hint information of the container is retained (cached) on the client end together with the node information (partition).

Communication overheads are kept to a minimum as the node maintaining the container is connected and processed directly without having to access the cluster to search for nodes that have been placed every time the container used by the application is switched.

Although the container placement changes dynamically due to the rebalancing process in GridDB, the position of the container is transmitted as the client cache is updated regularly. For example, even when there is a node mishit during access from a client due to a failure or a discrepancy between the regular update timing and re-balancing timing, relocated information is automatically acquired to continue with the process.

4.7.1 TQL and SQL

TQL in NoSQL products and SQL-92 compliant SQL in NewSQL products are supported as database access languages.

- What is TQL

  A simplified SQL prepared for NoSQL products. The support range is limited to functions such as search, aggregation, etc., using a container as a unit. TQL is employed by using the client API (Java, C language) of NoSQL products.

- What is SQL?

  Standardization of the language specifications is carried out in ISO to support the interface for defining and performing data operations in conformance with SQL-92 in GridDB. SQL uses the ODBC/JDBC of the new SQL product.

See “GridDB API Reference” (GridDB_API_Reference.html) for details on TQL, and “GridDB/NewSQL DB SQL Reference” (GridDB_NewSQL_SQL_Reference.pdf) for details on SQL.

4.7.2 API

Characteristic functions among the API provided by GridDB are explained. An interface to quickly process event information that occurs occasionally is available in NoSQL

When a large volume of events is sent to the database server every time an event occurs, the load on the network increases and system throughput does not increase. Significant impact will appear especially when the communication line bandwidth is narrow. Multi-processing is available in NoSQL to process multiple row registrations

for multiple containers and multiple inquiries (TQL) to multiple containers with a single request. The overall throughput of the system rises as the database server is not accessed frequently.

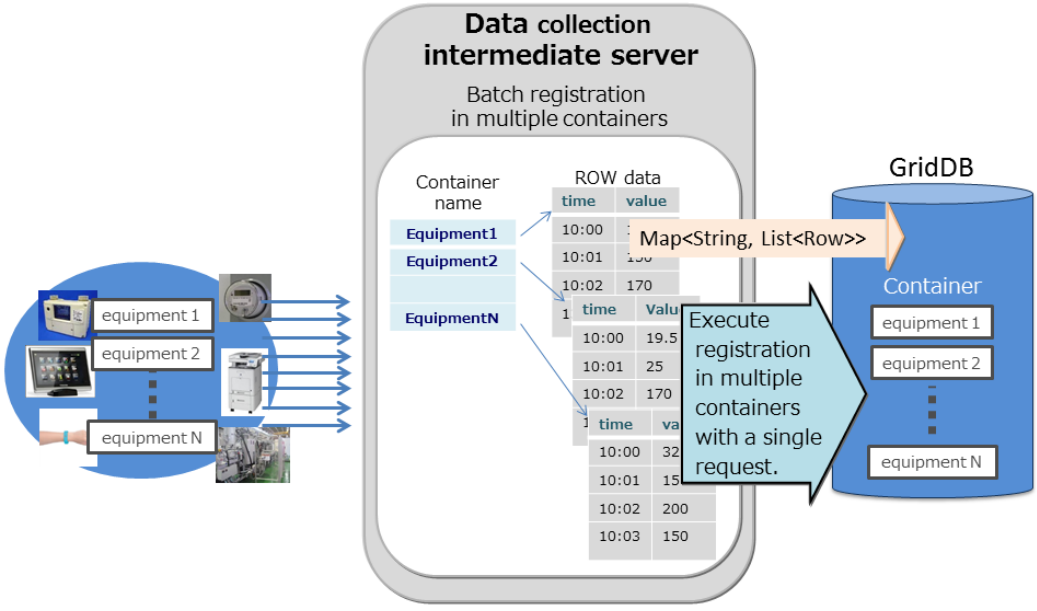
An example is given below.

- Multiput

  A container is prepared for each sensor name as a process to register event information from multiple sensors in the database. The sensor name and row array of the timeseries event of the sensor are created and a list (map) summarizing the data for multiple sensors is created. This list data is registered in the GridDB database each time the API is invoked.

  In the API of a multi- registration process, the communication process is optimized by consolidating requests for 1 or more containers to a node in GridDB formed by multiple clusters. In addition, multi-registrations are processed quickly without performing MVCC when executing a transaction.

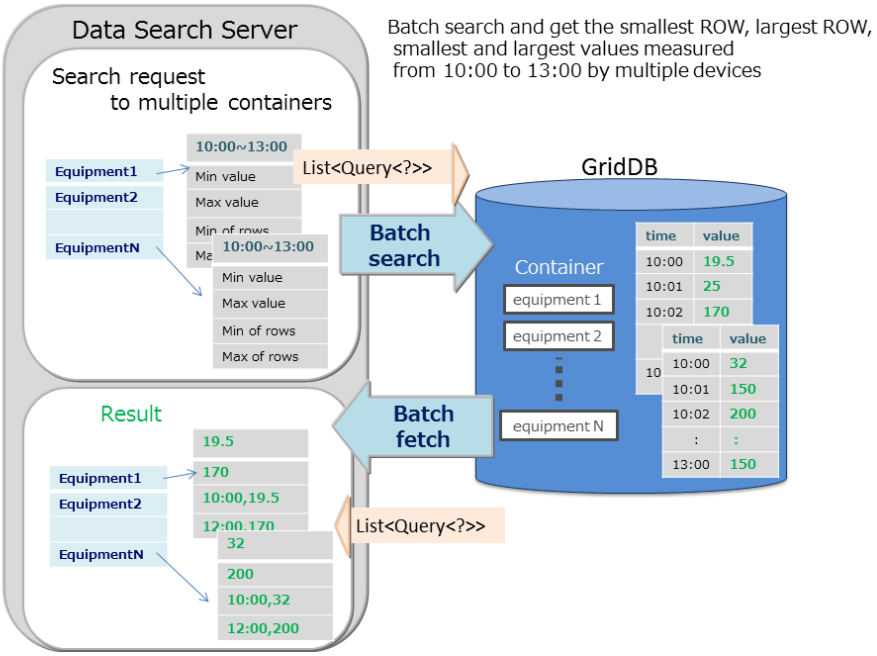
  In a multiput, transactions are committed automatically. Data is confirmed on a single case basis.



Multiput process

- Multi-query (fetchAll)

  Instead of executing multiple queries to a sensor, these can be executed in a single query by consolidating event information of the sensor. For example, this is most suitable for acquiring aggregate results such as the daily maximum, minimum and average values of data acquired from a sensor, or data of a row set having the maximum or minimum value, or data of a row set meeting the specified condition.



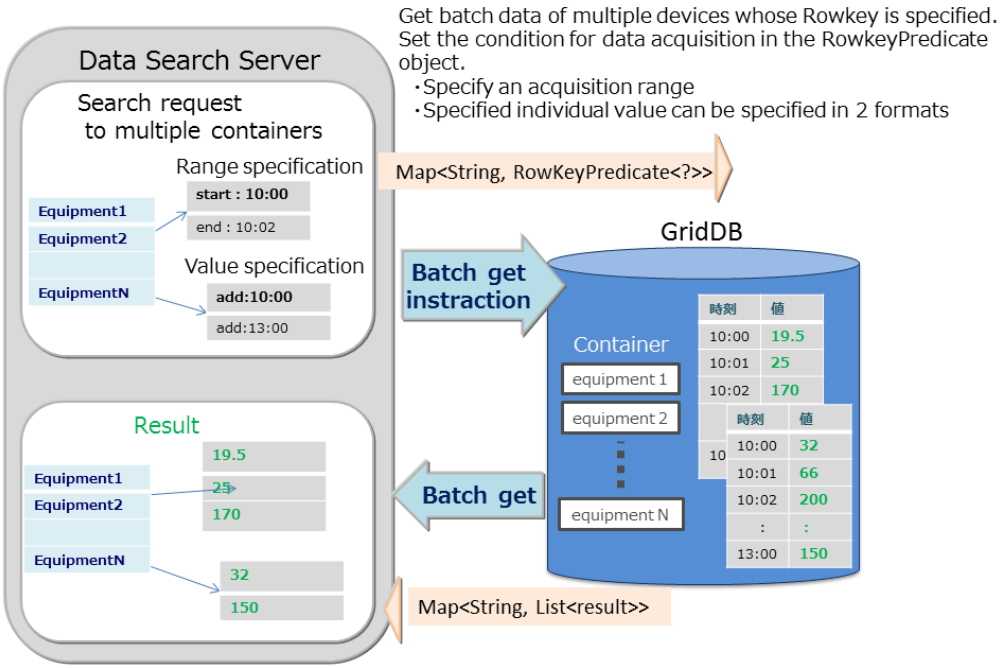
fetchAll process

- Multiget

  Batch data of multiple devices with the specified Rowkey can be acquired using a process to acquire event information of the sensor and so on. Set the condition for data acquisition in the RowkeyPredicate object to acquire data from multiple devices together.

  In a RowKeyPredicate object, the acquisition condition is set in either one of the 2 formats below.

  - Specify the acquisition range
  - Specified individual value



Multiget process

4.8 Operating function

GridDB has the following operating functions. This section provides an overview of the functions. See “GridDB Operating Management Guide” (GridDB_OperationGuide.html) for details on the operating functions.

5 parameters

Describes the parameters to control the operations in GridDB. In the GridDB parameters, there is a node definition file to configure settings such as the setting information and usable resources etc., and a cluster definition file to configure operational settings of a cluster. Explains the meanings of the item names in the definition file and the settings and parameters in the initial state.

The unit of the setting is set as shown below.

- The byte size can be specified in the following units: TB, GB, MB, KB, B, T, G, M, K, or lowercase notations of these units Unit cannot be omitted unless otherwise stated.
- Time can be specified in the following units: h, min, s, ms. Unit cannot be omitted unless otherwise stated.

5.1 Cluster definition file(gs_cluster.json)

The same setting in the cluster definition file needs to be made in all the nodes constituting the cluster. As the partitionNum and storeBlockSize parameters are important parameters to determine the database structure, they cannot be changed when GridDB is started after the system is built.

The cluster name is a parameter that must be set from V2.7 onwards.

The meanings of the various settings in the cluster definition file are explained below.

By adding an item name, items that are not included in the initial state can be recognized by the system. Indicate whether the parameter can be changed and the change timing in the change field.

- Change disallowed: Node cannot be changed once it has been started. The database needs to be initialized if you want to change the setting.
- Start: Parameter can be changed by restarting all the nodes constituting the cluster.

- Online: Parameters that are currently in operation online can be changed. However, the contents in the definition file need to be manual amended as the change details will not be perpetuated.

Configuration of GridDB	Initial value	Meaning of parameters and limitation values	change
/notificationAddress	239.0.0.1	Standard setting of a multi-cast address. This setting will become valid if a parameter with the same cluster, transaction name is omitted. If a different value is set, the address of the individual setting is valid.	restart
/dataStore /partitionNum	128	Specify a common multiple that will allow the number of partitions to be divided and placed by the number of constituting clusters Integer: Specify an integer that is 1 or higher and 10000 or lower.	Change disallowed
/dataStore /storeBlockSize	64KB	Specify the disk I/O size. Either 64KB or 1MB can be specified. Assume and set the data occurrence frequency. Cannot be changed after server is started.	Change disallowed
/cluster /clusterName	NIL	Specify the name for identifying a cluster. Mandatory input parameter.	restart
/cluster /replicationNum	2	No. of replicas. Partition is doubled if the no. of replicas is 2.	restart
/cluster /notificationAddress	239.0.0.1	Multi-cast address for cluster configuration	restart
/cluster /notificationPort	20000	Specify a value within a specifiable range as a multi-cast port no. for a cluster configuration.	restart
/cluster /notificationInterval	5sec	Specify a multi-cast period of 1s or more, or 2^31s or less for the cluster configuration.	restart
/cluster /heartbeatInterval	5sec	Specify a check period (heart beat period) of 1s or more, and less than 2^31s, to check the node survival among clusters.	restart

Configuration of GridDB	Initial value	Meaning of parameters and limitation values	change
/cluster /loadbalanceCheckInterval	180sec	In order to adjust the load balance among nodes constituting the cluster, specify a data sampling period of 1s or more, and less than 2^31s, with the unit omitted when determining whether to implement the balancing process or not.	restart
/sync /timeoutInterval	30sec	Timeout time during data synchronization among clusters If a timeout occurs, the system load may be high, or a failure may have occurred. Specify a value that is 1s or higher and less than 2^31s.	restart
/transaction /notificationAddress	239.0.0.1	Multi-cast address that a client connects to initially. Master node is notified in the client.	restart
/transaction /notificationPort	31999	Multi-cast port that a client connects to initially. Specify a value that is 1s or higher and less than 2^31s.	restart
/transaction /notificationInterval	5sec	Multi-cast period for a master to notify its clients. Specify a value that is 1s or higher and less than 2^31s.	restart
/transaction /replicationMode	0	Specify the data synchronization (replication) method when updating the data in a transaction. Specify a string or integer, "ASYNC"or 0 (non-synchronous), "SEMISYNC"or 1 (quasi-synchronous).	restart
/transaction /replicationTimeoutInterval	10 sec	Specify the timeout time for communications among nodes when synchronizing data in a quasi-synchronous replication transaction. Specify a value that is 1s or higher and less than 2^31s.	restart
/sql /notificationAddress	239.0.0.1	Multi-cast address when the JDBC/ODBC client is connected initially. Master node is notified in the client.	restart
/sql /notificationPort	41999	Multi-cast port when the JDBC/ODBC client is connected initially. Specify a value that is 1s or higher and less than 2^31s.	restart
/sql /notificationInterval	5 sec	Multi-cast period for a master to notify its JDBC/ODBC clients. Specify a value that is 1s or higher and less than 2^31s.	restart

5.2 Node definition file (gs_node.json)

Default setting of the resources in nodes constituting a cluster. In an online operation, there are also parameters whose values can be changed online from the resource, access frequency, etc., that have been laid out. Conversely, note that there are also values (concurrency) that cannot be changed once set.

The meanings of the various settings in the node definition file are explained below.

By adding an item name, items that are not included in the initial state can be recognized by the system. Indicate whether the parameter can be changed and the change timing in the change field.

- Change disallowed: Node cannot be changed once it has been started. The database needs to be initialized if you want to change the setting.
- Start: Parameter can be changed by restarting all the nodes constituting the cluster.
- Online: Parameters that are currently in operation online can be changed. However, the contents in the definition file need to be manual amended as the change details will not be perpetuated.

Specify the directory by specifying the full path or a relative path from the GS_HOME environmental variable. For relative path, the initial directory of GS_HOME serves as a reference point. Initial configuration directory of GS_HOME is /var/lib/gridstore.

Configuration of GridDB	Initial value	Meaning of parameters and limitation values	change
/serviceAddress	NIL	Set the initial value of each cluster, transaction, sync service address. The initial value of each service address can be set by setting this address only without having to set the addresses of the 3 items.	restart
/dataStore /dbPath	data	The deployment directory of the database file is specified by the full path or a relative path	restart

Configuration of GridDB	Initial value	Meaning of parameters and limitation values	change
/dataStore /backupPath	backup	Specify the backup file deployment directory path.	restart
/dataStore /storeMemoryLimit	1024MB	Upper memory limit for data management	online
/dataStore /concurrency	1	Concurrency	Change disallowed
/dataStore /logWriteMode	1	If the log writing mode period is -1 or 0, log writing is performed at the end of the transaction. If it is 1 or more and less than 2^31, log writing is performed at a period specified in seconds	restart
/dataStore /persistencyMode	1(NORMAL)	In the perpetuation mode, the period that the update log file is maintained during a data update is specified. Specify either 1 (NORMAL) or 2 (RETAINING_ALL_LOGS). For "NORMAL", a transaction log file which is no longer required will be deleted by the checkpoint. For "RETAINING_ALL_LOGS", all transaction log files are retained. Default value is "1 (NORMAL)".	restart
/dataStore /storeWarmStart	true (valid)	Specify whether to save in-memory up to the upper limit of the chunk memory during a restart.	restart
/dataStore /affinityGroupSize	4	Number of affinity groups	restart
/checkpoint /checkpointInterval	1200 sec	Checkpoint process execution period to perpetuate a data update block in the memory	restart
/checkpoint /checkpointMemoryLimit	1024MB	Upper limit of special checkpoint write memory* Pool the required memory space up to the upper limit when there is a update transaction in the checkpoint.	online
/checkpoint /useParallelMode	false (invalid)	Specify whether to execute the checkpoint concurrently. *The no. of concurrent threads is the same as the concurrency.	restart

Configuration of GridDB	Initial value	Meaning of parameters and limitation values	change
/checkpoint /checkpointCopyInterval	100ms	Output process interval when outputting a block with added or updated data to a disk in a checkpoint process.	restart
/cluster /serviceAddress	Follow the higher order "serviceAddress"	Standby address for cluster configuration	restart
/cluster /servicePort	10010	Standby port for cluster configuration	restart
/sync /serviceAddress	Follow the higher order "serviceAddress"	Specify the reception address for data synchronization among the clusters.	restart
/sync /servicePort	10020	Standby port for data synchronization	restart
/system /servicePort	Follow the higher order "serviceAddress"	Standby address for REST command	restart
/system /eventLogPath	10040	Standby port for REST command	restart
/system /eventLogPath	log	Event log file deployment directory path	restart
/transaction /serviceAddress	Follow the higher order "serviceAddress"	Standby address for transaction process	restart
/transaction /servicePort	10001	Standby port for transaction process	restart
/transaction /connectionLimit	5000	Upper limit of the no. of transaction process connections	restart
/transaction /transactionTimeoutLimit	0 sec	Transaction timeout upper limit. Timeout does not occur at 0 sec	restart

Configuration of GridDB	Initial value	Meaning of parameters and limitation values	change
/sql /servicePort	210001	Standby port for New SQL access process	restart
/sql /connectionLimit	5000	Upper limit of the no. of connections processed for New SQL access	restart
/sql /concurrency	5	No. of simultaneous execution threads	restart

6. Terminology

Describes the terms used in GridDB in a list.

Terms	Meaning
Node	Refers to the individual server process to perform data management in GridDB.
Cluster	Refers to a single or a set of multiple nodes to perform data management together.
Master node	Node to perform a cluster management process.
Follower node	A node participating in a cluster except the master node.
Number of nodes constituting a cluster	Refers to the number of nodes constituting a GridDB cluster. When starting GridDB for the first time, the number is used as a threshold value for the cluster to be valid. (Cluster service is started when the number of nodes constituting a cluster joins the cluster.)
Number of nodes already participating in a cluster	Number of nodes currently in operation that have been incorporated into the cluster among the nodes constituting the GridDB cluster.
Block	A block is a data unit for data perpetuation in a disk (hereinafter known as a checkpoint) and is the smallest physical data management unit in GridDB. Multiple container data is placed in a block. Before initial startup of GridDB, a size of either 64 KB or 1 MB can be selected for the block size to be set up in the definition file (cluster definition file). Specify 64 KB if the installed memory of the system is low, or if the frequency of data increase is low.
Partition	Data management unit to arrange a container. Smallest data placement unit between clusters and data movement and replication unit for adjusting the load balance between nodes (rebalance) and for managing the data multiplexing (replica) in case of a failure.

Terms	Meaning
Partition group	A group summarizing multiple partitions which is equivalent to the data file in the file system when the data is perpetuated in a disk. 1 checkpoint file corresponds to 1 partition group. Partition groups are created according to concurrency (/dataStore/concurrency) figure in the node definition file.
Row	Refers to 1 row of data registered in a container or table. Multiple rows are registered in a container or table. Columns of multiple data type are created in a row.
Container	Data structure serving as an I/F with the user. Container to manage a set of rows. 2 data types exist, collection and timeseries container.
Collection	One type of container to manage rows having a general key.
Timeseries container	One type of container to manage rows having a timeseries key. Possesses a special function to handle timeseries data.
Table	A table is a special container form that exists only in NewSQL products. SQL can be operated as an interface in NewSQL products.
Database file	A perpetuated file group to write data saved in a node that constitutes a cluster into a disk or SSD. A database file is a generic term to describe the transaction log file that is saved every time the GridDB database is updated and the checkpoint file that is written regularly by the database in the memory.
Checkpoint file	A partition group is a file written into a disk. Update information is reflected in the memory by a cycle of the node definition file (/checkpoint/checkpointInterval).
Transaction log file	Transaction update information is save sequentially as a log.
LSN (Log Sequence Number)	Shows the update log sequence no. when updating in the transaction assigned to each partition. The master node of a cluster configuration contains the maximum LSD (MAXLSN) of all the partitions maintained by each node.

Terms	Meaning
Replica	Refers to the multiplexing placement of partitions in multiple nodes. A replica may be an owner replica which is master data to be updated or a backup replica used for reference purposes.
Owner node	A node that can update a container in a partition. A node that records the container serving as a master among the replicated containers.
Backup node	A node that records the container serving as a replica among the replicated containers.
Definition file	There are 2 types of definition file, a parameter file (gs_cluster.json: hereinafter known as a cluster definition file) when composing a cluster, and a parameter file (gs_node.json: hereinafter known as a node definition file) to set the operations and resources of the node in the cluster. In addition, there is also a user definition file for GridDB administrator users.
Event log file	The operating log of the GridDB server is saved. Messages such as errors, warnings, etc. are saved.
OS user (gsadm)	A user known as gsadm is created during GridDB installation and who has the right to execute operating functions in GridDB
Administrator user	An administrator user is a GridDB use prepared to perform operations in GridDB.
General user	A user used in the application system.
User definition file	File in which an administrator user is registered. During initial installation, 2 administrators, system and admin, are registered.
Cluster database	General term for all databases that can be accessed in a GridDB cluster system.
Database	Theoretical data management unit created in a cluster database. A public database is created in a cluster database by default. Data separation can be realized for each user by creating a new database and giving a general user the right to use it.

Terms	Meaning
Full backup	A backup of the cluster database currently in use is stored online in the backup directory specified in the node definition file.
Incremental backup (Cumulative/Differential backup)	A backup of the cluster database currently in use is stored online in the backup directory specified in the node definition file. In subsequent backups, only the difference in the update block after the backup is backed up.
Auto log backup	In addition to backing up the cluster database currently in use in the specified directory online, the transaction log is also automatically picked up at the same timing as the transaction log file writing. The write timing of the transaction log file follows the value of /dataStore/logWriteMode in the node definition file.
Failover	When a failure occurs in a cluster currently in operation, the structure allows the backup node to automatically take over the function and continue with the processing.
Client failover	When a failure occurs in a cluster currently in operation, the structure allows the backup node to be automatically re-connected to continue with the processing as a retry process when a failure occurs in the API on the client side.
Table partitioning	Function to access a huge table quickly by allowing concurrent execution by processors of multiple nodes, and the memory of multiple nodes to be used effectively by distributing the placement of a large amount of table data with multiple data registrations in multiple nodes.
Data affinity	A function to raise the memory hit rate by placing highly correlated data in a container in the same block and localizing data access.
Node affinity	A function to reduce the network load when accessing data by placing highly correlated containers and tables in the same node.

7 System limiting values

Block size	64KB	1MB
String/spatial data size	31KB	128KB
BLOB data size	127MB	1GB
Array length	4000	65000
No. of columns	1024	1024
No. of columns subject to linear complementary compression	100	100
Size of container name	About 16KB	About 128KB
Size of column	256Byte	256Byte
Partition size	About 64TB	About 1PB
Cluster name	64 characters	64charactors
General user name	64 characters	64 characters
Database name	64 characters	64 characters
Password	64 characters	64 characters
No. of users	128	128
No. of databases	128	128
Size of trigger name	256Byte	256Byte
URL of trigger	4KB	4KB
Number of affinity groups	10000	10000
Length of data affinity string	8 characters	8 characters

Block size	64KB	1MB
No. of divisions in a timeseries container with a cancellation deadline	160	160
Size of communication buffer managed by a GridDB node	Approximately 2 GB	Approximately 2 GB

- String, container name, column name, trigger name, URL of trigger
    - Limiting value is equivalent to UTF-8 encode