東芝が開発したオープンソースのIoT向けデータベースGridDB
～最新の強化ポイントについて～

TOSHIBA

東芝デジタルソリューションズ株式会社　GridDBコミュニティ版担当
野々村 克彦
2024.10.18

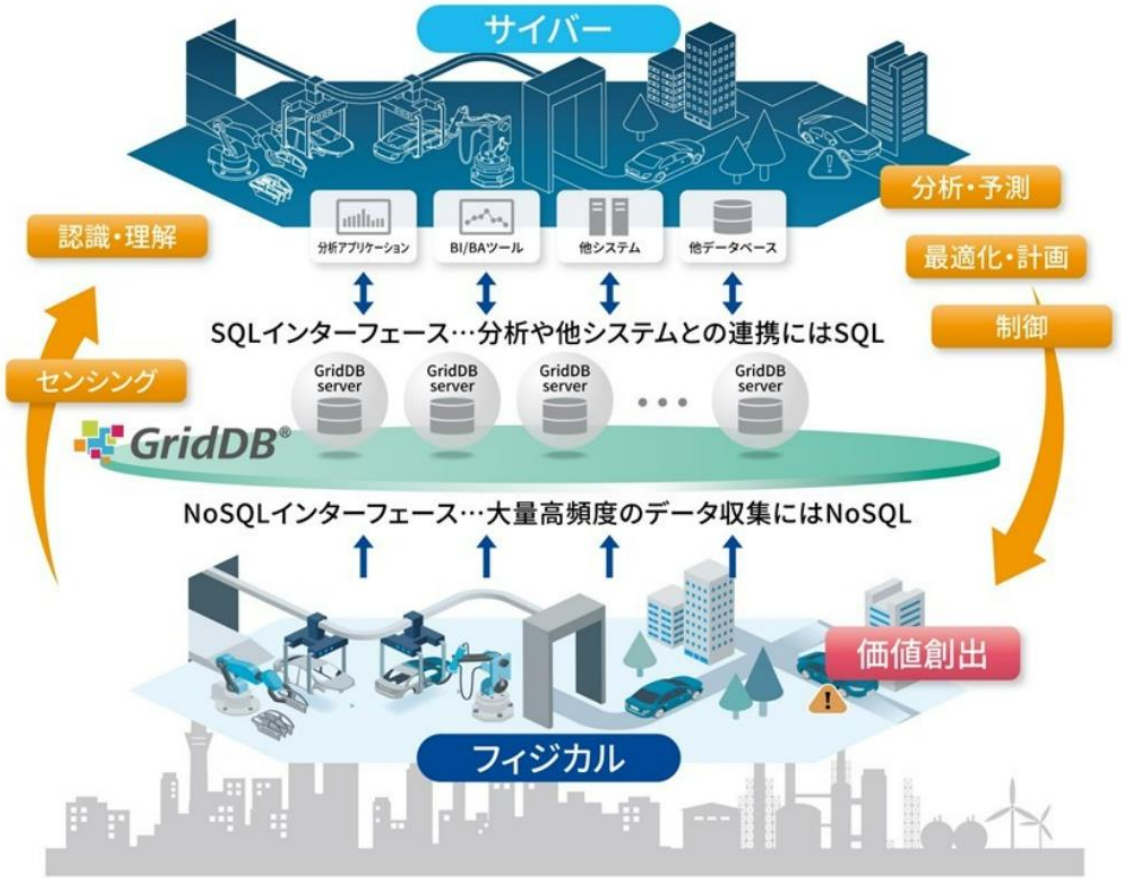
Contents

01

GridDBの概要

GridDBについて

- GridDBは、東芝デジタルソリューションズが開発した
  日本発のNoSQL型のDBMSです。
- ビッグデータやIoTシステム向けに特化して作られたDBMSです。



主な適用事例

・電力会社 低圧託送業務システム

　スマートメータから収集される電力使用量を集計し、需要量と発電量のバランスを調整

https://www.global.toshiba/jp/company/digitalsolution/articles/tsoul/22/004.html

・HDD製造会社 品質管理システム

　製造装置のセンサーデータを長期に渡って蓄積・分析し、品質分析・改善に適用

https://www.global.toshiba/content/dam/toshiba/jp/products-solutions/ai-iot/griddb/event/2024/%E8%B3%87%E6%96%99_A20_session_TDSL_GridDB.pdf

・ものづくりIoTソリューション「Meister Factoryシリーズ」

　IoT データの蓄積を担う「Meister DigitalTwin」のデータ基盤として利用

https://www.global.toshiba/jp/products-solutions/manufacturing-ict/meister-factory.html

社会インフラ、製造業を中心に、高い信頼性・可用性が求められるシステムに適用されています。

ラインアップ

GridDB
Community Edition

高頻度・大量に発生するデータの蓄積とリアルタイムな活用をスムーズに実現する次世代のオープンソースデータベース

GridDB
Enterprise Edition

高頻度・大量に発生する時系列データの蓄積とリアルタイムな活用をスムーズに実現し、ビジネスを大きく成長させるために最適化された次世代のデータベース

2024/10/15
V5.7リリース

GridDB Cloud

高頻度・大量に発生する時系列データの蓄積とリアルタイムな活用をスムーズに実現するクラウドデータベースサービス

2024/10/15
V2.5リリース

https://www.global.toshiba/jp/news/digitalsolution/2024/10/news-20241015-01.html

オープンソースのIoTデータベース GridDB CE

- **GitHub上にソース公開**
  - https://github.com/griddb/griddb
- **オープンソース化の目的**
  - ビッグデータ技術の普及促進
    - 多くの人に知ってもらいたい、使ってみてもらいたい。
    - いろんなニーズをつかみたい。
  - 他のオープンソースソフトウェア、システムとの連携強化

- V2.8 (2016年)
  NoSQL機能をGitHub上にソース公開
- V4.5 (2020年)
  SQL機能もソース公開
- 最新版 V5.6 (2024年5月)

V5.7は2024年11月にソース公開予定

GridDB CEの特徴

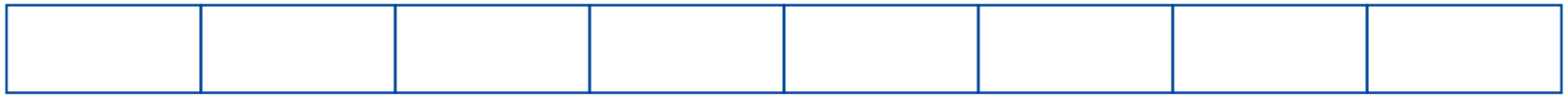
時系列データ指向	• データモデルはキー・コンテナ。コンテナ内でのデーター貫性を保証 • 巨大テーブルに対するインターバル（ハッシュ）パーティショニング • パーティショニング期限解放、分析関数(SQL)
開発の俊敏性と使いやすさ	• NoSQL（キーバリュー型）インタフェースだけではなく、SQLインタフェースを提供（デュアルインタフェース） • （SQLインタフェース）ジョインなど複数テーブルに対するSQL
高い処理能力	• メモリを主、ストレージを従としたハイブリッド型インメモリDB • （SQLインタフェース）SQLにおける分散並列処理 • （NoSQLインタフェース）バッチ処理 MultiPut/MultiGet/MultiQuery
拡張性	• ペタバイト級の大規模データへの対応 • コアスケールへの対応

※　チェックポイント、Redoログによる耐障害性にも対応しています

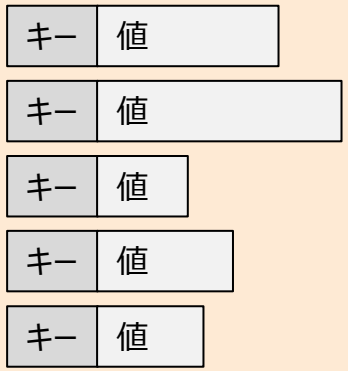
NoSQL DB (Key Value Store(KVS))とキー・コンテナモデル

<キー、バリュー>

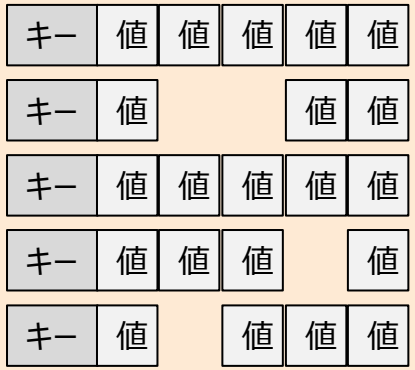
ハッシュテーブル



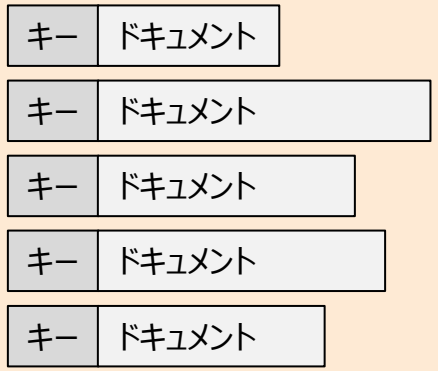
キーバリュー型
（例：Redis）



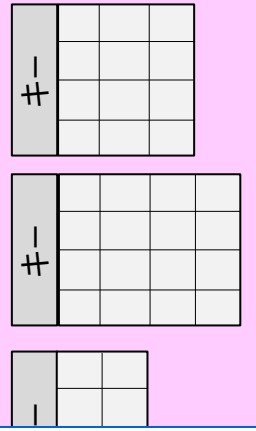
カラム指向型
（例：Cassandra）



ドキュメント指向型
（例：MongoDB）

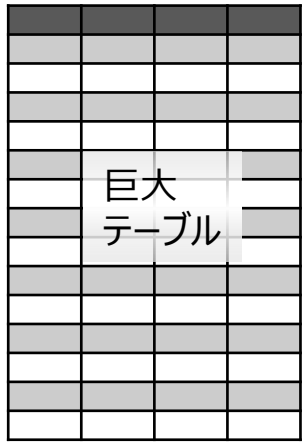


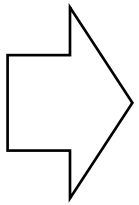
キーコンテナ型
GridDB

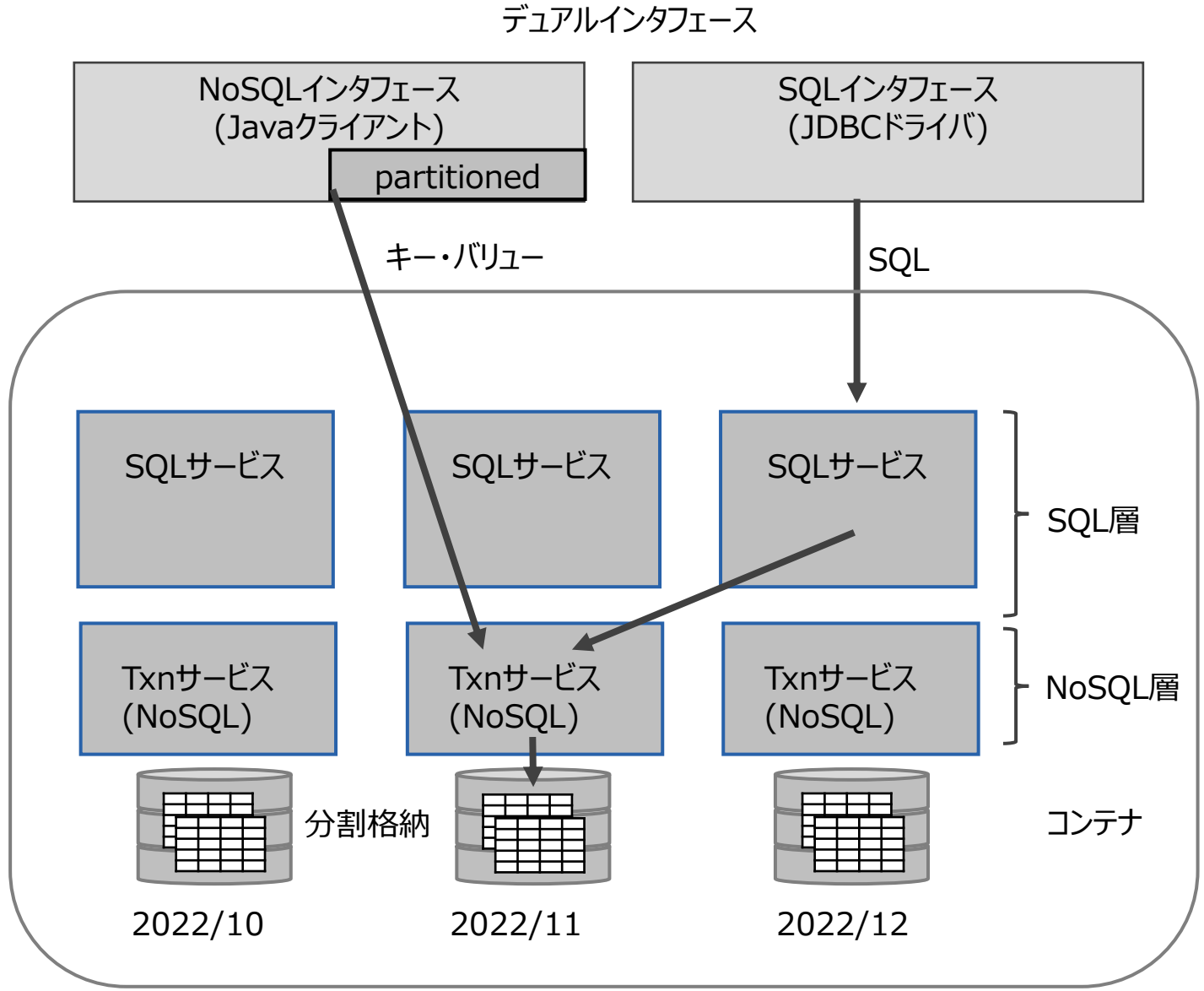


※コンテナ（テーブル）名がキーになる
※索引、検索言語TQL、トランザクションをサポート

デュアルインタフェースとテーブルパーティショニング



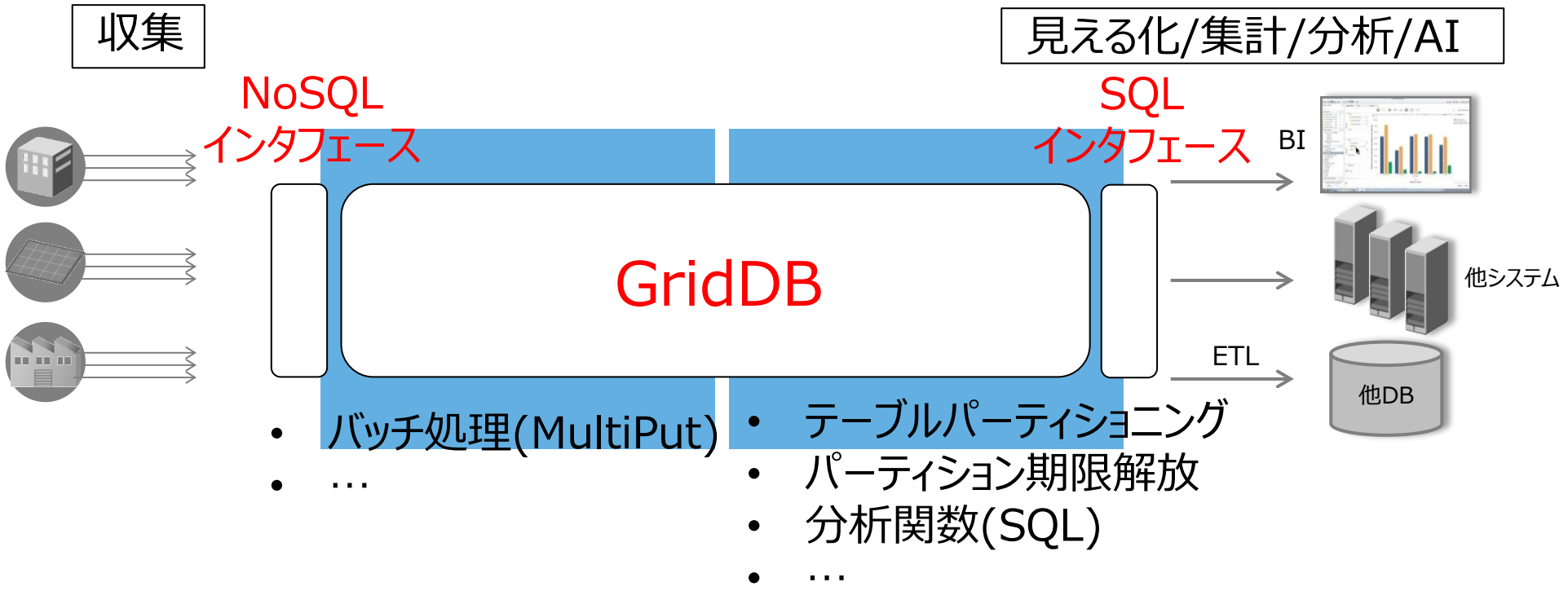




テーブルパーティショニング
- インターバルパーティショニング
- インターバルハッシュパーティショニング

など

NoSQL/SQLデュアルインタフェースによるシステム化



- NoSQL＋SQLによる高速処理
- SQLインタフェースによる他システム連携強化

02

GridDBの最新の強化ポイント

最新の強化ポイントの一覧

- 2024/2 GridDB V5.5CE
- 2024/5 GridDB V5.6CE
- 2024/9 GridDB Kafkaコネクタ　V0.6

	V5.5CE	V5.6CE
機能強化（主に時系列機能）	①JDBCバッチ更新 （addBatch関数など）	②時系列データ自動集計
性能強化（主に大規模対応）	③コストベースによるジョイン順序決定	④データベース圧縮アルゴリズムの追加 ⑤テーブルパーティショニングの強化 （時間単位の分割指定）
その他	⑥Zabbix向け監視テンプレート	

機能強化

① JDBCバッチ更新

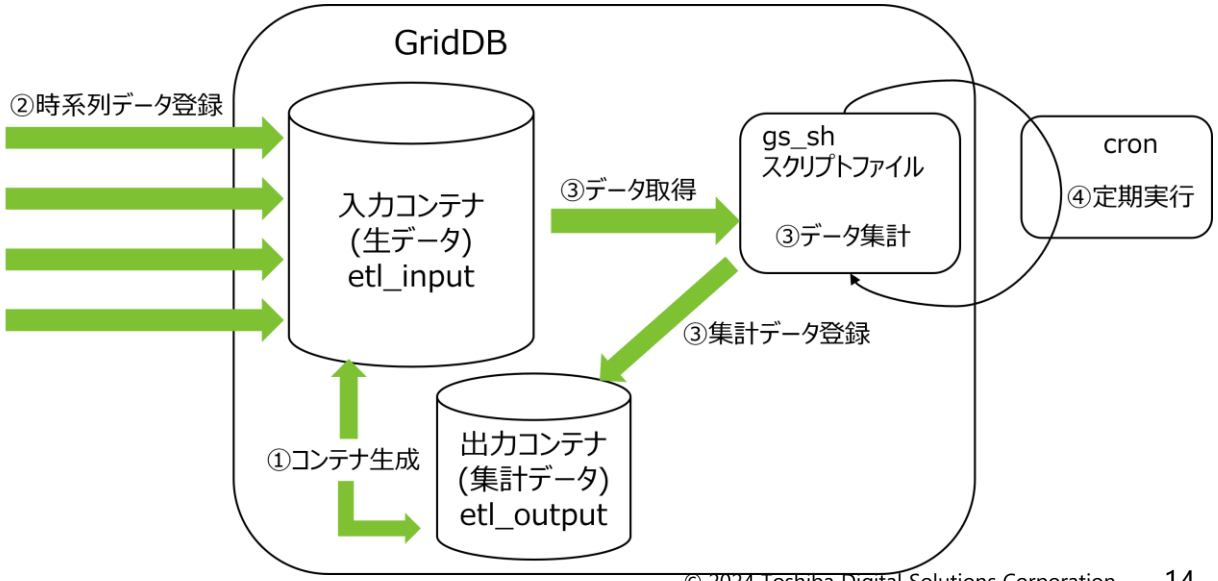
バッチ更新を行うJDBCメソッド(addBatchなど)を新たにサポートしました。

② 時系列データ自動集計

蓄積された大量の時系列データを自動的に集計して集計結果を蓄積しておくことで、ユーザが集計結果を取得する際に、処理時間を短縮できるようになります。

今回、GridDBコマンドラインインタフェース(gs_sh)を使って自動集計が実現できるようになりました。

```
// (1)プリペアードステートメント作成
String sql = "insert into SampleJDBC_AddBatch (id, value) Values(?,?)";
PreparedStatement pstmt = con.prepareStatement(sql);
// (2)値を設定する
pstmt.setInt(1,1);
pstmt.setString(2,"test0");
pstmt.addBatch();
pstmt.setInt(1,2);
pstmt.setString(2,"test1");
pstmt.addBatch();
// (3)SQL実行
int[] cnt = pstmt.executeBatch();
```



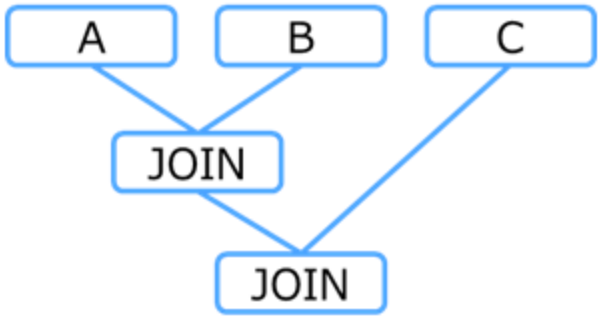
性能強化 (1/2)

③ コストベースによるジョイン順序決定

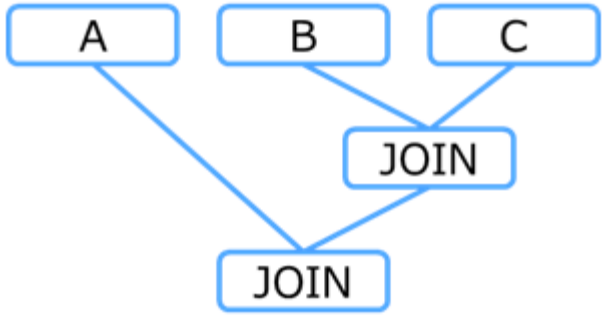
検索実行の過程で求められるロウ数をコストとして計算し、複数テーブルの最適なジョイン順序を決定してプランを生成できるようになりました。

```
SELECT * FROM A,B,C
WHERE A.x = B.x AND B.y = C.y
```

ルールベースの場合



コストベースの場合



テーブル	ロウ件数
A	1,000,000
B	10,000
C	10,000

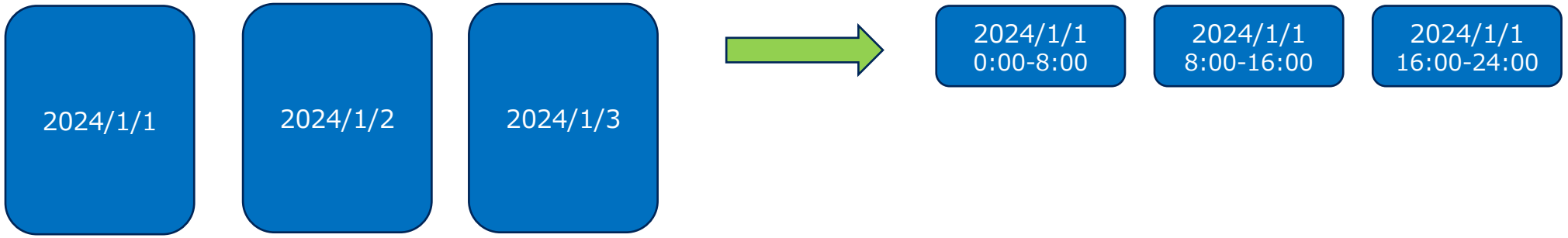
性能強化 (2/2)

④ データベース圧縮アルゴリズムの追加

- ノード定義ファイル(gs_node.json)のdataStore/storeCompressionModeで設定
  - ZLIB圧縮（従来）："COMPRESSION_ZLIB" または "COMPRESSION"
  - ZSTD（Standard）圧縮：“COMPRESSION_ZSTD”。ZLIB圧縮より圧縮解凍速度や圧縮率で優れています。

⑤ テーブルパーティショニングの強化（時間単位の分割指定）

- 1日当たりのデータ規模が大きくなるケース向けに、1時間(HOUR)単位でテーブル分割できるようになりました。



その他

⑥ Zabbix向け監視テンプレート
GridDBの監視を行うためのZabbix向けのテンプレートをサンプルとして
追加しました。

「GridDB向けの主なデータ」

- Disk space usage（/var/lib/gridstoreのディスク利用量）
- Store memory usage（ストアメモリ利用量）
- StoreDetail.storeMemory（ストアメモリ利用量の詳細）
  - metaData：メタデータ
  - rowData：ロウデータ
  - mapData：索引データ
  - batchFreeRowData：期限付きのロウデータ
  - batchFreeMapData：期限付きの索引データ

ご参考： Zabbix向け監視テンプレートによる画面表示の例

Disk space usage

Store memory usage

StoreDetail.storeMemory

03

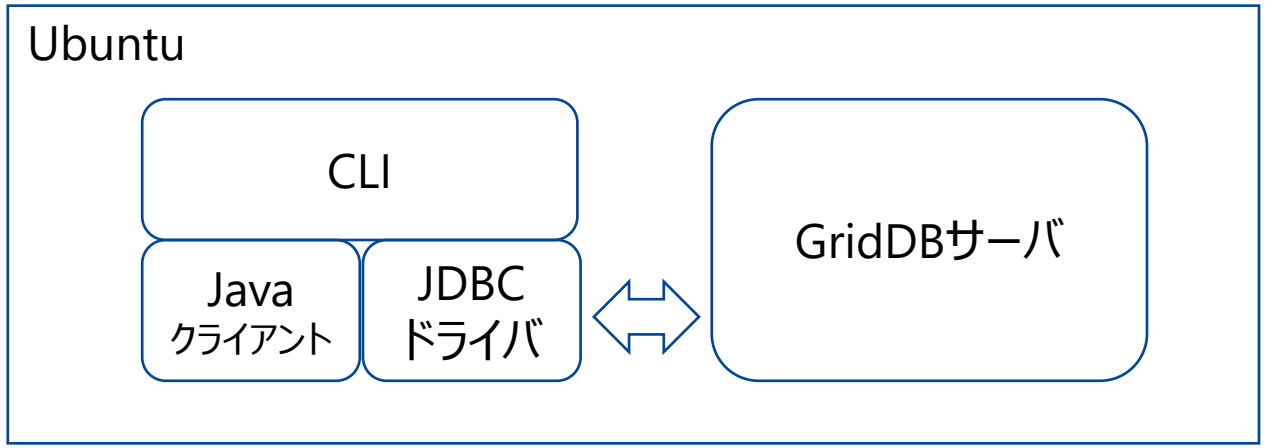
GridDBの利用方法

- GridDBのインストール・実行
- GridDB Kafkaコネクタのインストール・実行

動作環境の例

＜動作環境＞
- Windows 11上にWSL2 Ubuntu 22.04。Java 11(デフォルト)インストール済
- メモリは16GB（ブラウザ、Zoom等のアプリを動かさないのであれば8GBも可）

＜前提条件＞
- 同一マシンに全ソフトウェアをインストール。ローカル実行
- GridDBのクラスタ名はmyCluster(デフォルト)
- GridDB管理者の名前はadmin、パスワードはadmin



※GridDBサーバ、Javaクライアント：https://github.com/griddb/griddb
※GridDB JDBCドライバ： https://github.com/griddb/jdbc
※GridDB CLI：https://github.com/griddb/cli

GridDBのインストール＆起動の手順

【インストール】

1. GridDBサーバのインストール

```
$ wget https://github.com/griddb/griddb/releases/download/v5.6.0/griddb_5.6.0_amd64.deb
$ sudo dpkg -i griddb_5.6.0_amd64.deb
```

2. GridDB CLI（コマンドライン・インタフェース）のインストール

```
$ wget https://github.com/griddb/cli/releases/download/v5.6.0/griddb-ce-cli_5.6.0_amd64.deb
$ sudo dpkg -i griddb-ce-cli_5.6.0_amd64.deb
```

【起動】

3. GridDBのサービス起動

```
$ sudo systemctl start gridstore
```

4. CLI起動

```
$ sudo su - gsadm
$ gs_sh
```

[SQLの例]

```
# テーブル作成
> create table t1 (c0 long, c1 long);
# データ登録
> insert into t1 values(1, 2);
# 検索
> select * from t1;
> get
```

※GridDBサービスの停止

```
$ systemctl stop gridstore
```

※ サービス機能(systemctl)が利用できない環境の場合は、gs_startnode等の運用コマンドをご利用ください。

GridDBのインストール＆起動の手順

【インストール】

1. GridDBサーバのインストール

```
$ wget https://github.com/griddb/griddb/releases/download/v5.6.0/griddb_5.6.0_amd64.deb
$ sudo dpkg -i griddb_5.6.0_amd64.deb
```

2. GridDB CLI（コマンドライン・インタフェース）のインストール

```
$ wget https://github.com/griddb/cli/releases/download/v5.6.0/griddb-ce-cli_5.6.0_amd64.deb
$ sudo dpkg -i griddb-ce-cli_5.6.0_amd64.deb
```

【起動】

3. GridDBのサービス起動

```
$ sudo systemctl start gridstore
```

4. CLI起動

```
$ sudo su - gsadm
$ gs_sh
```

※GridDBサービスの停止

```
$ systemctl stop gridstore
```

[SQLの例]

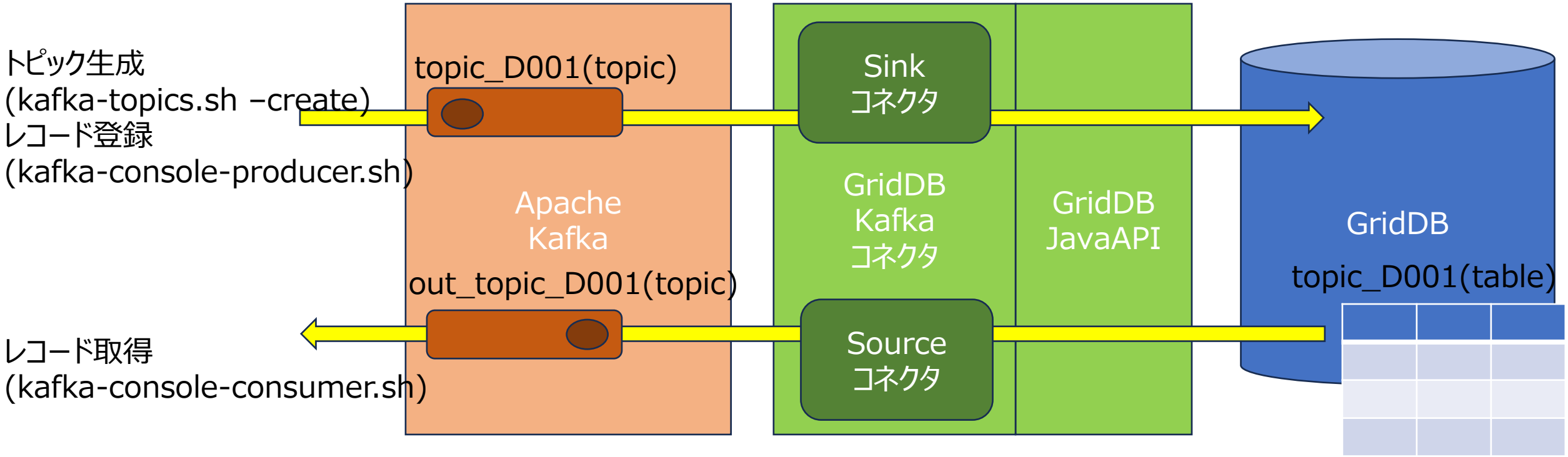
```
# テーブル作成
> create table t1 (c0 long, c1 long);
# データ登録
> insert into t1 values(1, 2);
# 検索
> select * from t1;
> get
```

わずかなステップだけで
CLIによるSQLなどの操作が開始できます。

※ サービス機能(systemctl)が利用できない環境の場合は、gs_startnode等の運用コマンドをご利用ください。

Apache KafkaとGridDB Kafkaコネクタ(https://github.com/griddb/griddb-kafka-connect)

- Apache Kafka：オープンソースの分散イベント ストリーミング プラットフォーム
  https://kafka.apache.org/
  利用例：センサデータをトピックと呼ばれるキューに保持しておき、リアルタイム処理やDBへの登録を実現
- GridDB Kafkaコネクタ： ２つのコネクタから構成されます
  - Sinkコネクタ：KafkaトピックからDBへの登録
  - Sourceコネクタ：DBからKafkaトピックへの登録



Kafkaのインストール・起動

※ KAFKAHOME：Kafkaをインストールしたフォルダ

[端末①]

Kafkaインストール：

$ wget https://dlcdn.apache.org/kafka/3.7.1/kafka_2.13-3.7.1.tgz

$ tar xzvf kafka_2.13-3.7.1.tgz

$ cd kafka_2.13-3.7.1

Kafkaサーバ起動：

$ export PATH=$PATH:<KAFKAHOME>/bin

$ zookeeper-server-start.sh -daemon config/zookeeper.properties

$ kafka-server-start.sh config/server.properties

GridDB Kafkaコネクタのインストール・起動

※ KAFKAHOME：Kafkaをインストールしたフォルダ

[端末②]

GridDB Kafkaコネクタインストール：

$ git clone https://github.com/griddb/griddb-kafka-connect

$ cd griddb-kafka-connect

$ mvn clean install

- 生成されたtarget/griddb-kafka-connector-X.X.jarを
  <KAFKAHOME>/kafka_2.13-3.7.1/libs/にコピー
- config/griddb-sink.propertiesとconfig/griddb-source.propertiesを
  <KAFKAHOME>/kafka_2.13-3.7.1/config/にコピー
- コピーしたgriddb-sink.propertiesファイルと
  griddb-source.propertiesファイルを編集

GridDB Kafkaコネクタ起動：※デモの都合上、次のスライドの２つのタイミングで

$ export PATH=$PATH:<KAFKAHOME>/bin

$ cd <KAFKAHOME>

$ connect-standalone.sh
  config/connect-standalone.properties
  config/griddb-sink.properties
  config/griddb-source.properties

[griddb-sink.properties]

```
host=127.0.0.1
port=10001
cluster.name=myCluster
user=admin
password=admin
container.type=TIME_SERIES
topics.regex=topic(.*)
transforms=TimestampConverter
transforms.TimestampConverter.type=org.apache.kafka.connect.
transforms.TimestampConverter$Value
transforms.TimestampConverter.format=yyyy-MM-dd hh:mm:ss
transforms.TimestampConverter.field=datetime
transforms.TimestampConverter.target.type=Timestamp
```

[griddb-source.properties]

```
host=127.0.0.1
port=10001
cluster.name=myCluster
user=admin
password=admin
containers=topic_D001
mode=timestamp
timestamp.column.name=datetime
topic.prefix=out_
```

Kafka経由でGridDBへのデータ登録・GridDBからのデータ取得について

※ KAFKAHOME：Kafkaをインストールしたフォルダ
※ GKCONHOME：GridDB Kafkaコネクタをインストールしたフォルダ

[端末③]

$ export PATH=$PATH:<KAFKAHOME>/bin

$ cd <GKCONHOME>/docs/sample/sink

状態確認(0)

$ kafka-topics.sh --list --bootstrap-server localhost:9092 # トピック一覧表示

$ kafka-console-consumer.sh --topic topic_D001 --from-beginning --bootstrap-server localhost:9092 # (topic_D001)

→ トピックtopic_D001が作成される

- [端末②] GridDB Kafkaコネクタを起動

$ ./script_sink.sh # rawdata.txtを入力にデータ登録

[rawdata.txt] 現在時刻が2024-10-18 15:00:00 JSTとした場合、UTCは06:00:00

```
2024-10-18 06:00:00.000000  D001  Ignore  Ignore  1  Control4-Door
2024-10-18 06:01:00.000000  D001  Ignore  Ignore  2  Control4-Door
2024-10-18 06:02:00.000000  D001  Ignore  Ignore  3  Control4-Door
```

状態確認(1)

$ kafka-console-consumer.sh --topic D001 --from-beginning --bootstrap-server localhost:9092

→ { “payload”: …} # (topic_D001) payloadで始まるデータが登録件数分表示されます

- GridDB側にテーブルtopic_D001とデータが登録されます [GridDBのCLIを使って状態確認]
- [端末②] GridDB Kafkaコネクタを再起動

$ kafka-console-consumer.sh --topic out_topic_D001 --from-beginning --bootstrap-server localhost:9092

→ { “schema”: … “messge”:”1”…} # (out_topic_D001) schemaで始まる。現在時刻までのデータが表示されます

しばらく経過後に状態確認(2)

$ kafka-console-consumer.sh --topic out_topic_D001 --from-beginning --bootstrap-server localhost:9092

→ { “schema”: … “messge”:”1”…}

{ “schema”: … “messge”:”2”…}

(out_topic_D001) schemaで始まる。現在時刻までのデータが表示されます。件数が増えます

理由：以下の検索でデータを取得しているため。
tql = select * where datetime > 前回取得日時 and datetime < 現在時刻

ご参考：データ登録用に利用したscript_sink.sh

```
#!/bin/bash
function echo_payload {
    echo '{ "payload": {  "datetime": "'$1 $2'",  "sensor": "'$3'",  "translate01": "'$4'",  "translate02": "'$5'",  "message": "'$6'",  "sensoractivity": "'$7'" }, "schema": {  "fields": [  {  "field": "datetime",  "optional": false,  "type": "string"  },  {  "field": "sensor",  "optional": false,  "type": "string"  },  {  "field": "translate01",  "optional": false,  "type": "string"  },  {  "field": "translate02",  "optional": false,  "type": "string"  },  {  "field": "message",  "optional": false,  "type": "string"  },  {  "field": "sensoractivity",  "optional": false,  "type": "string"  }  ],  "name": "sample",  "optional": false,  "type": "struct" }}
'}


TOPICS=()


for file in `find $1 -name ¥*rawdata.txt` ; do
    echo $file
    LOCATION="topic"
    head -10 $file |while read -r line ; do
        SENSOR=`echo ${line} | awk '{ print $3 }'`
        if [[ ! " ${TOPICS[@]} " =~ " ${LOCATION}_${SENSOR} " ]]; then
            echo Creating topic ${LOCATION}_${SENSOR}
            kafka-topics.sh --bootstrap-server 127.0.0.1:9092 --create --topic  ${LOCATION}_${SENSOR} 2&>1 /dev/null
            TOPICS+=(${LOCATION}_${SENSOR})
        fi
        echo_payload ${line} | kafka-console-producer.sh --topic ${LOCATION}_${SENSOR} --bootstrap-server localhost:9092
    done
done
```

ご参考：Kafkaスクリプト

スクリプト名	オプション	機能
kafka-topics.sh	--create	トピックの作成
	--list	トピック一覧を取得
	--describe	トピック詳細を取得
	--delete	トピックの削除
kafka-console-producer.sh		データをトピックに登録
kafka-console-consumer.sh		トピックからデータを取得

ご参考：関連ブログ

[GridDB Kafkaコネクタ(JavaAPI)]

- GridDBとKafkaでデータをストリームする

https://griddb.net/ja/blog/stream-data-with-griddb-and-kafka/

[GridDB Kafkaコネクタ(JDBC)]

- GridDB v5.5、JDBC、KafkaでSQLバッチ挿入を使用する

https://griddb.net/ja/blog/using-sql-batch-inserts-with-griddb-v5-5-jdbc-and-kafka/

- KafkaとGridDBの連携

https://griddb.net/ja/blog/using-kafka-with-griddb/

- JDBCでKafkaのソースとしてGridDBを使用する

https://griddb.net/ja/blog/using-griddb-as-a-source-for-kafka-with-jdbc/

04

OSS活動

主なOSS活動

① GridDB本体の機能強化

② 主要OSSとの連携強化

③ APIの拡充

④ GitHub以外のサイトからの情報発信

- パッケージ
- デベロッパーズサイト（ホワイトペーパ、ブログなど）
- SNS

⑤ 主要OSSリポジトリへのコントリビュート

⑥ プラットフォームの拡充

⑦ その他

- OSCなどカンファレンス参加

OSS活動の全体イメージ

性能測定 | 収集 | 分散処理 | 可視化 | Webアプリ | 分析 | AI/機械学習 …

④GitHub以外のサイトからの情報発信
PyPI/npm/Maven/Packagist/…

②主要OSSとの連携強化

Spark コネクタ
Fluentd/Grafana/Redash プラグイン
YCSB コネクタ
Kafka コネクタ
Hadoop MapReduce コネクタ

⑥プラットフォームの拡充
Ubuntu、RockyLinux
Windows、MacOS
Docker

WebAPI
Python/Node.JS/Go/PHP/Ruby/Perl/Rustクライアント

③APIの拡充

Javaクライアント
JDBCドライバ
Cクライアント

①GridDB本体の機能強化
GridDB V5.6 CE(Community Edition)

https://github.com/griddb/
GitHub

⑤主要OSSリポジトリへのコントリビュート

デベロッパーズサイト

https://griddb.net/

griddb net	検索

- **アプリケーション開発者向けのサイト**
- **様々なコンテンツを公開**
  - ホワイトペーパ
  - ブログ

  など

X（旧ツイッター）

https://x.com/griddb_jp

twitter griddb	検索

- **GridDBに関するリリース、イベント、などをお知らせします。**
  (日本国内向け)

05

まとめ

まとめ

- GridDBはビッグデータ・IoT向けのデータベースです。
- GridDBの概要、最新の強化ポイント、GridDBの利用方法についてご紹介しました。
  - 今後も様々な拡張、拡充を進めて参ります。

GridDBのオープンソース版(GridDB CE)を是非とも使ってみてください。
https://github.com/griddb/
こんな使い方をしたい、こんな機能が欲しい等のご意見もお待ちしています。
（日本語でOKです）

TOSHIBA

各エディションの違い

- インタフェースはほぼ同じ
- クラスタ構成の有無の違い

項目	機能	Community Edition	Enterprise Edition	Cloud
サポート			✓	✓
プロフェッショナルサービス			✓	✓
データ管理	時系列コンテナ	✓	✓	✓
	コレクションコンテナ	✓	✓	✓
	索引	✓	✓	✓
	アフィニティ	✓	✓	✓
	テーブルパーティショニング	✓	✓	✓
クエリ言語	TQL	✓	✓	✓
	SQL	✓	✓	✓
NoSQLインタフェース	Java	✓	✓	✓
	C言語	✓	✓	✓
NewSQL(SQL) インタフェース	JDBC	✓	✓	✓
	ODBC		✓	✓
WebAPI		✓	✓	✓
時系列データ	時系列分析関数	✓	✓	✓
	期限付き解放機能	✓	✓	✓
クラスタリング	機能クラスタ構成		✓	✓
	分散データ管理		✓	✓
	レプリケーション		✓	✓
運用管理	ローリングアップグレード		✓	
	オンラインバックアップ		✓	✓
	エクスポート / インポート	✓	✓	✓
	運用管理GUI		✓	✓
	CLIツール	✓	✓	✓
セキュリティ	信暗号化 (TLS/SSL)		✓	✓
	認証機能 (LDAP)		✓	
オンプレミス環境	オンプレミス環境	✓	✓	
クラウドサービス	クラウドサービス			✓

ご参考：GridDBに関する情報

- **GridDB GitHubサイト**
  - https://github.com/griddb/griddb/
- **GridDB デベロッパーズサイト**
  - https://griddb.net/
- **X（旧Twitter）：GridDB (日本)**
  - https://x.com/griddb_jp
- **X（旧Twitter）：GridDB Community**
  - https://x.com/GridDBCommunity
- **Facebook: GridDB Community**
  - https://www.facebook.com/griddbcommunity/
- **Wiki**
  - https://ja.wikipedia.org/wiki/GridDB
- **GridDB お問い合わせ**
  - OSS版のプログラミング関連：Stackoverflow(https://ja.stackoverflow.com/search?q=griddb)もしくはGitHubサイトの各リポジトリのIssueをご利用ください
    プログラミング関連以外：contact@griddb.netもしくはcontact@griddb.orgをご利用ください

griddb github	検索

griddb net	検索

twitter griddb	検索