日本発オープンソース！！
スケールアウト型データベース GridDB入門
～ GitHubからダウンロードして使ってみましょう ～

TOSHIBA

東芝デジタルソリューションズ株式会社
野々村 克彦
2018.10.27

プロフィール

Katsuhiko Nonomura

knonomura

Block or report user

Organizations

名前：野々村 克彦

所属：東芝デジタルソリューションズ（株）ソフトウェア＆AIテクノロジーセンター　知識･メディア処理技術開発部

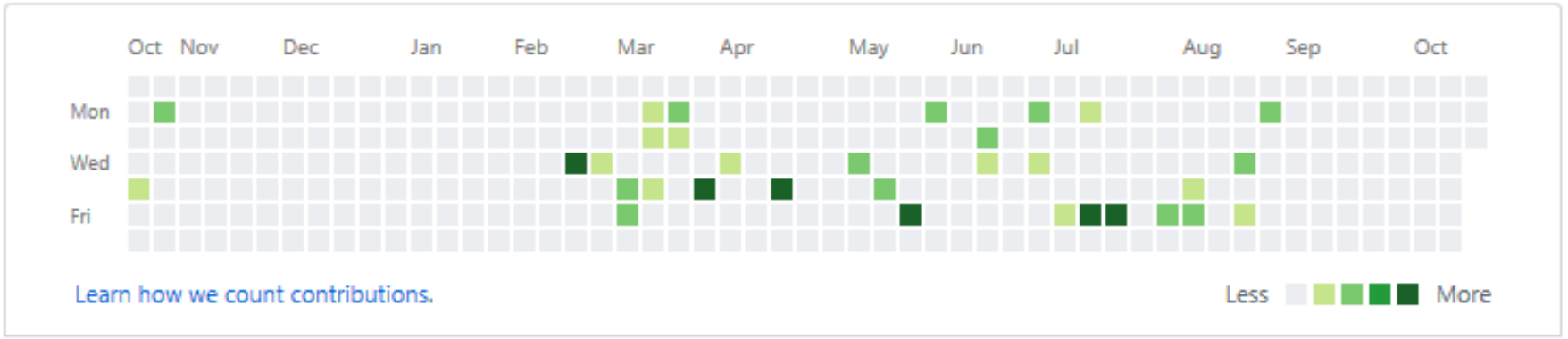
２０１１年　スケールアウト型DB GridDBの開発メンバ

２０１５年　GridDBのオープンソースPJ開始

現在 GridDBコミュニティ版の開発･コミッター、海外展開の技術支援など。

GitHub歴　４年

142 contributions in the last year



ちなみに、出身高校は米子東（鳥取県）。日本発のRuby考案者である、まつもとゆきひろ氏の２年後輩になるらしい。

発表内容

1. スケールアウト型データベースGridDBの概要

- 特長、導入事例

2. GridDBの使い方

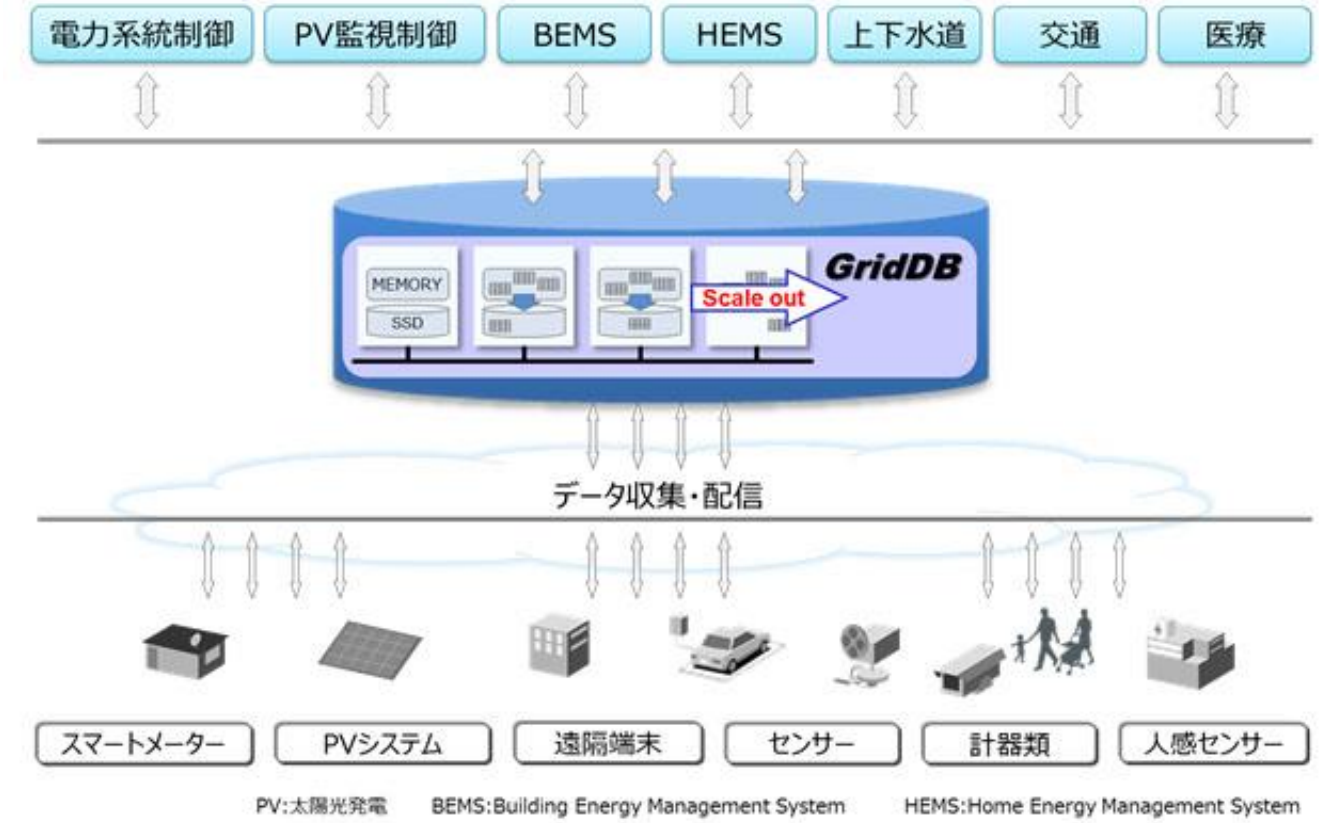
- クイックスタート (デモ)
- 運用コマンド操作、クライアント操作
- はまりやすいポイント、クラスタ構成やAWS/Azureでの利用

3. 最近のOSS活動

4. まとめ

GridDBとは

- 日本発のビッグデータ／IoT向けのスケールアウト型データベース
- 製品化（２０１３年）、基本機能をオープンソース化（２０１６年）
- 社会インフラを中心に、高い信頼性・可用性が求められるシステムで使われている



GridDBの特長

IoT指向のデータモデル

- 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
- 時系列データ管理する特別な機能

①キーコンテナ型

高い信頼性と可用性
High Availability

- データの複製をノード間で自動的に実行
- ノード障害があってもフェールオーバによりサービス継続
- 数秒から数十秒の切り替え時間

②自律データ再配置技術ADDA

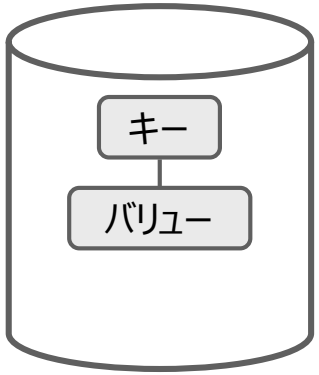
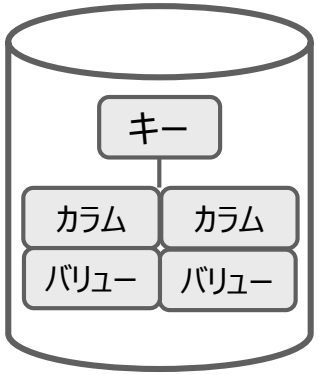
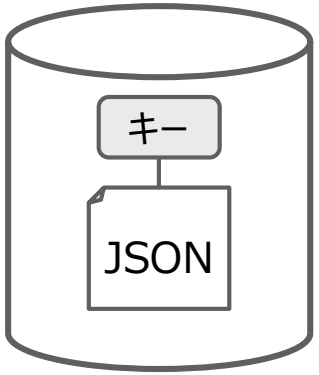
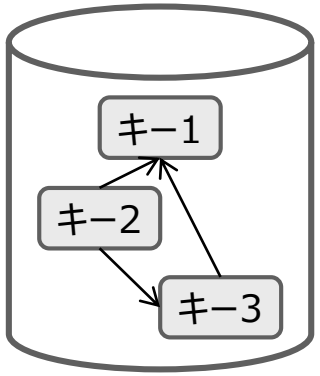
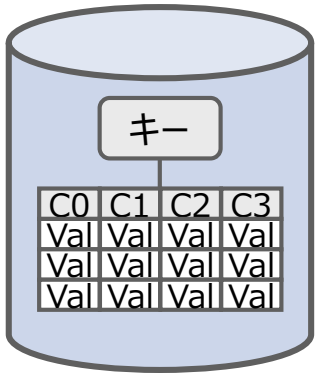
高いスケーラビリティ
High Scalability

- 少ないサーバ台数で初期投資を抑制
- 負荷や容量の増大に合わせたノード増設が可能
- 自律データ再配置により、高いスケーラビリティを実現

高性能
High Performance

- 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
- メモリやディスクの排他処理や同期待ちを極力排除

データモデル　①キーコンテナ型

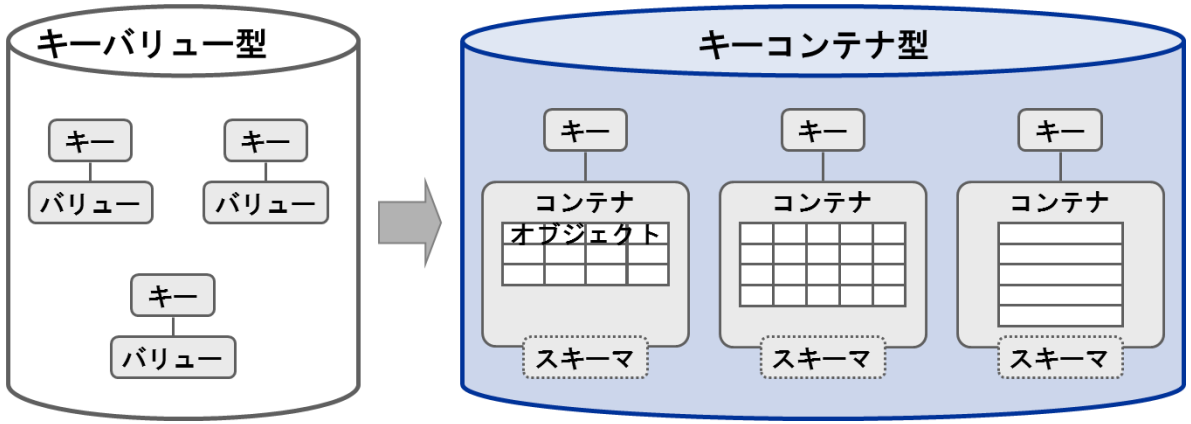
データモデル	キーバリュー型	ワイドカラム型	ドキュメント型	グラフ型	キーコンテナ型
					
NoSQLの例	Redis	Cassandra	MongoDB	Neo4j	GridDB

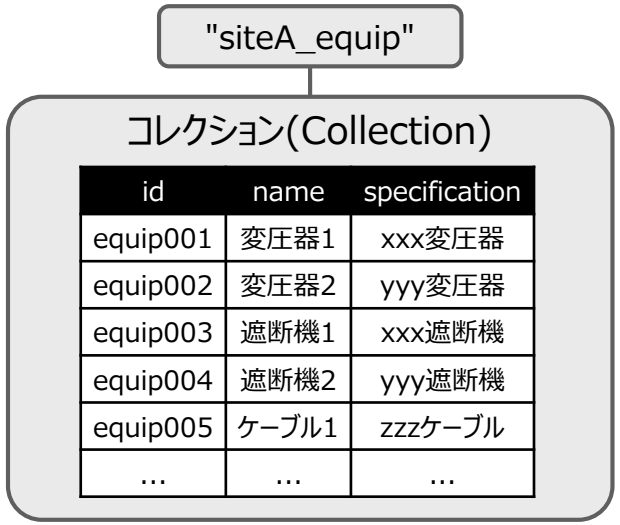
データモデル　①キーコンテナ型

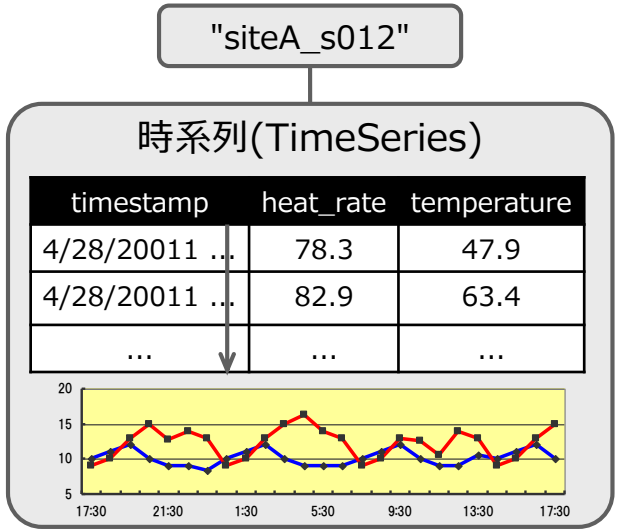
- **NoSQL型でよく採用されているキー・バリューを拡張**
- **順序に関係無くレコードが格納されるコレクションコンテナ**
- **時間順にレコードが格納される時系列コンテナ**

  時系列レコードを圧縮する機能や期限解放する機能
- **コンテナ内でのデーター貫性を保証**

  索引設定機能、SQLライクのクエリ(TQL)機能

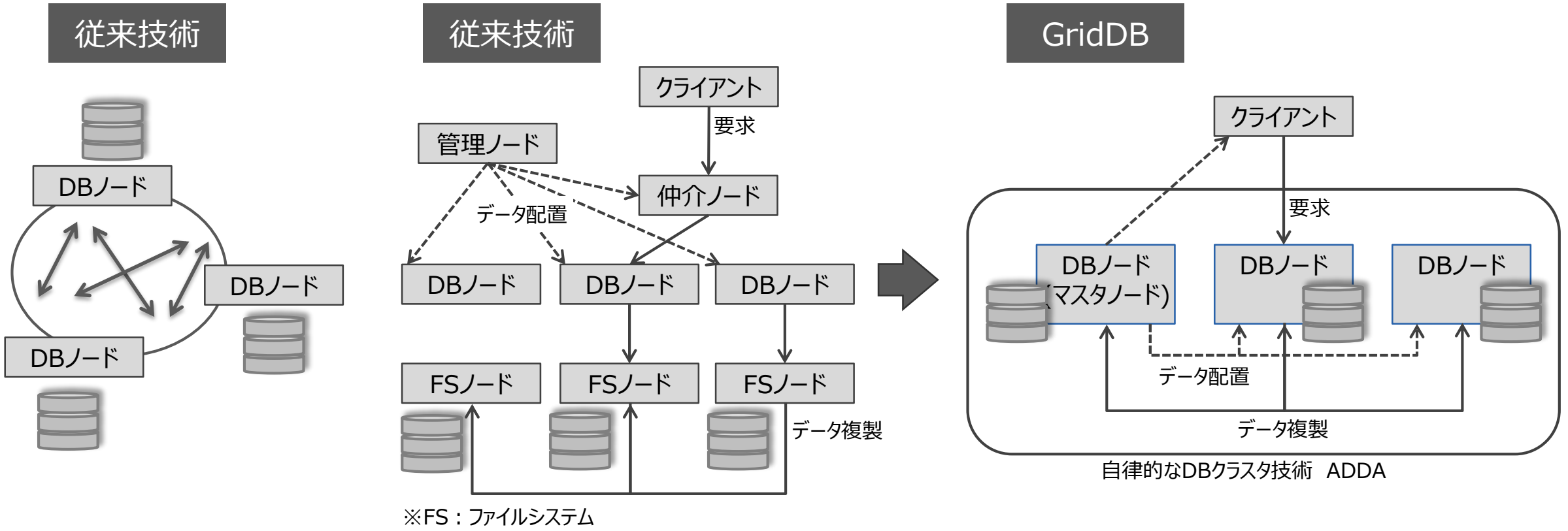






DBクラスタ　②ADDA

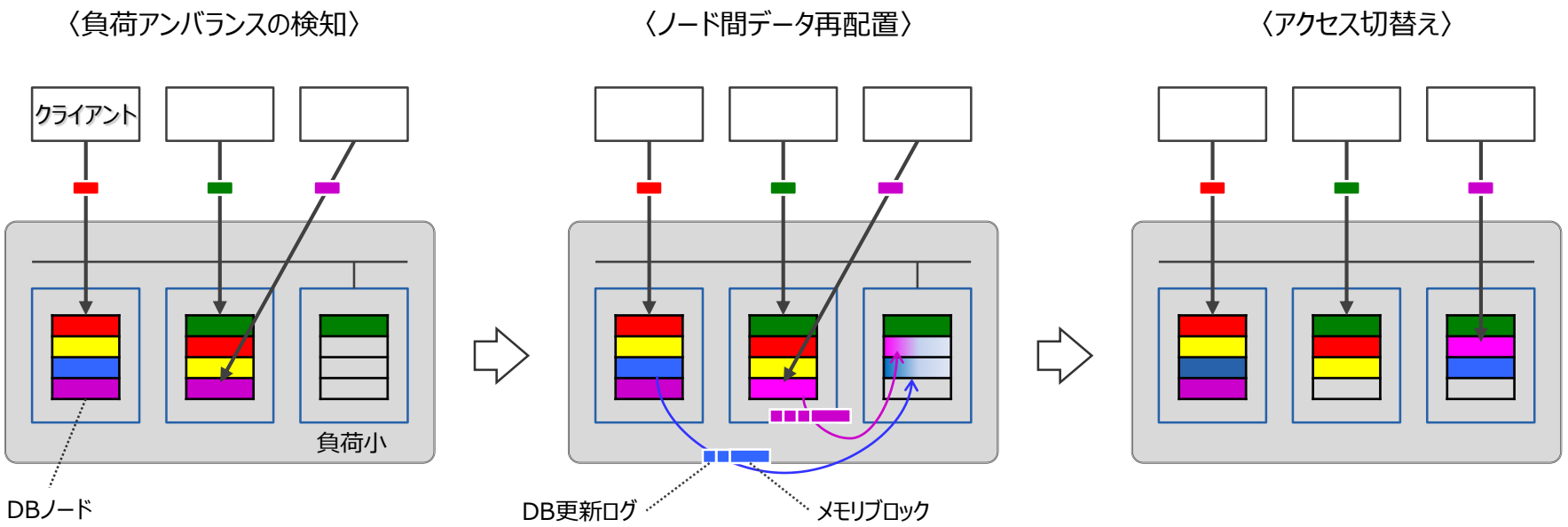
- 自律データ再配置技術(ADDA : Autonomous Data Distribution Algorithm)
- データを分散化するゆえにデータの一貫性が弱くなり、
  一貫性やスケール性を求めるとパフォーマンスが落ちる、という大きな欠点を解決



具体的な挙動

- **マスタースレーブモデルの改良**
  - ノード間でマスタノードを自動選択。管理ノードがクラスタ内に存在せず、単一障害点を完全排除

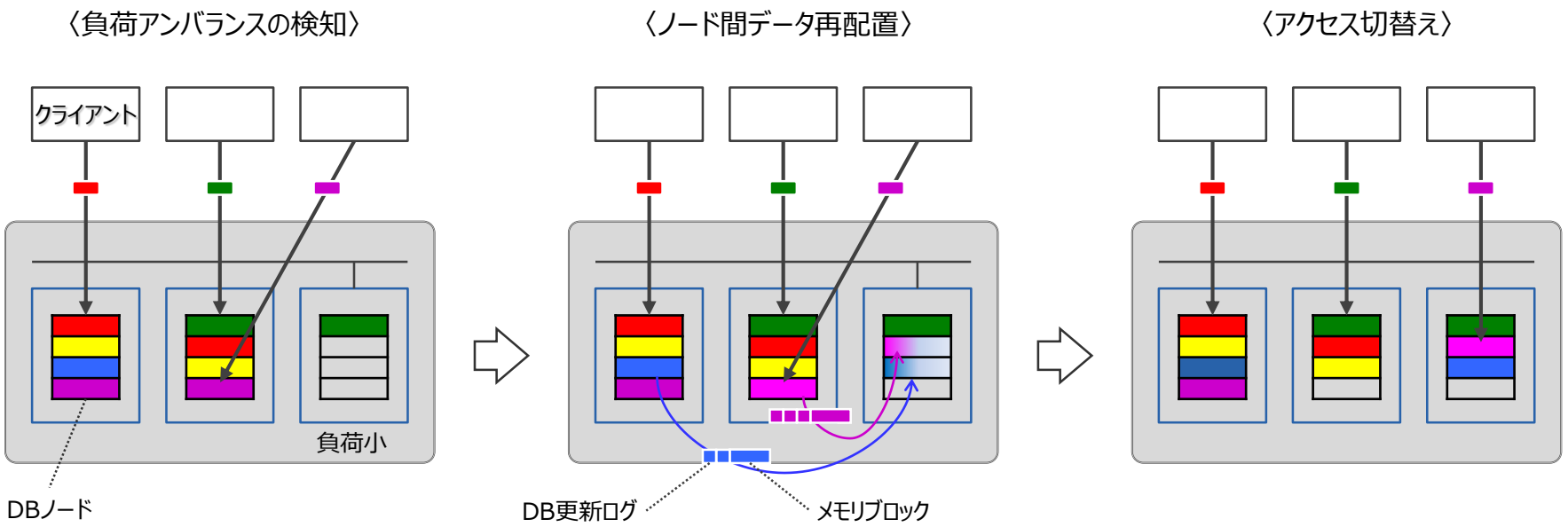
- **自律データ再配置技術の開発**
  - (マスターノードが)ノード間アンバランス、レプリカ欠損を検知⇒バックグラウンドでデータ再配置
  - 2種類のレプリカデータを使って高速同期、完了後切替え



具体的な挙動

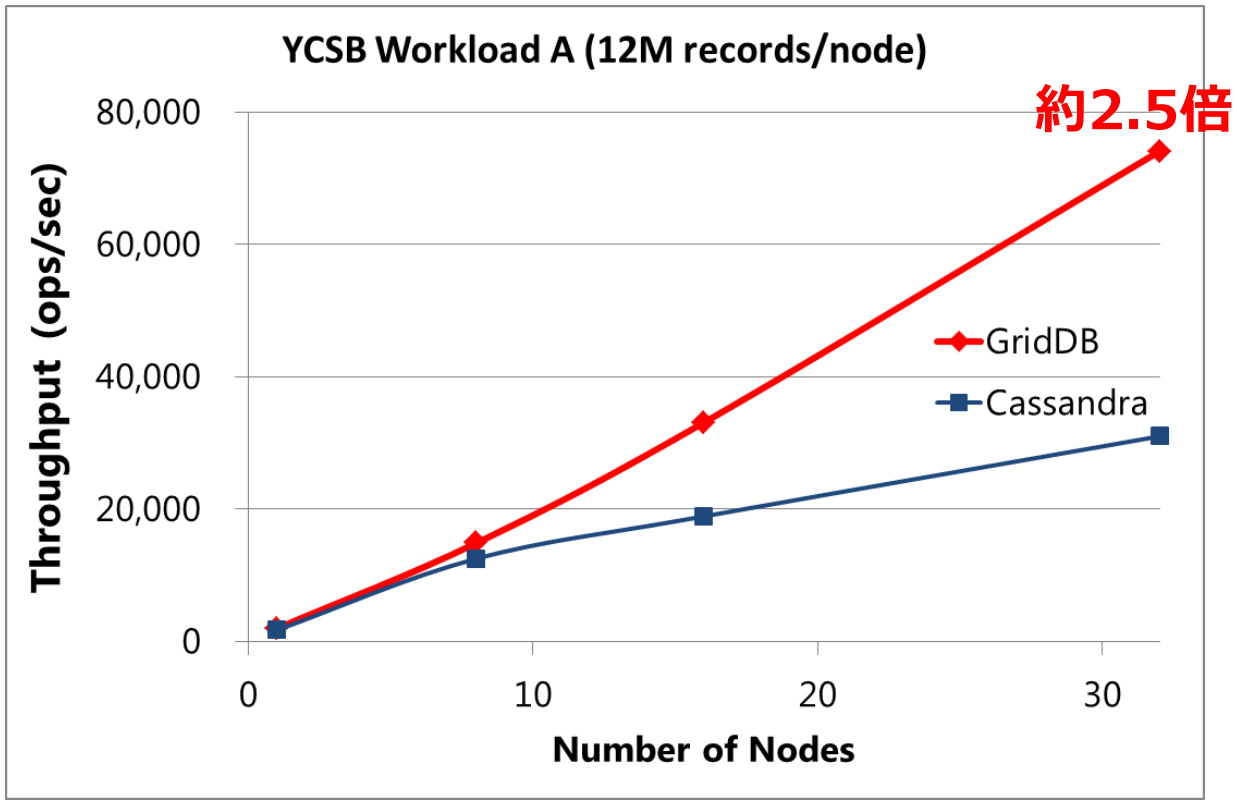
- **マスタースレーブモデルの改良**
  - ノード間でマスタノードを自動選択。管理ノードがクラスタ内に存在せず、単一障害点を完全排除

- **自律データ再配置技術の開発**
  - (マスターノードが)ノード間アンバランス、レプリカ欠損を検知⇒バックグラウンドでデータ再配置
  - 2種類のレプリカデータを使って高速同期、完了後切替え

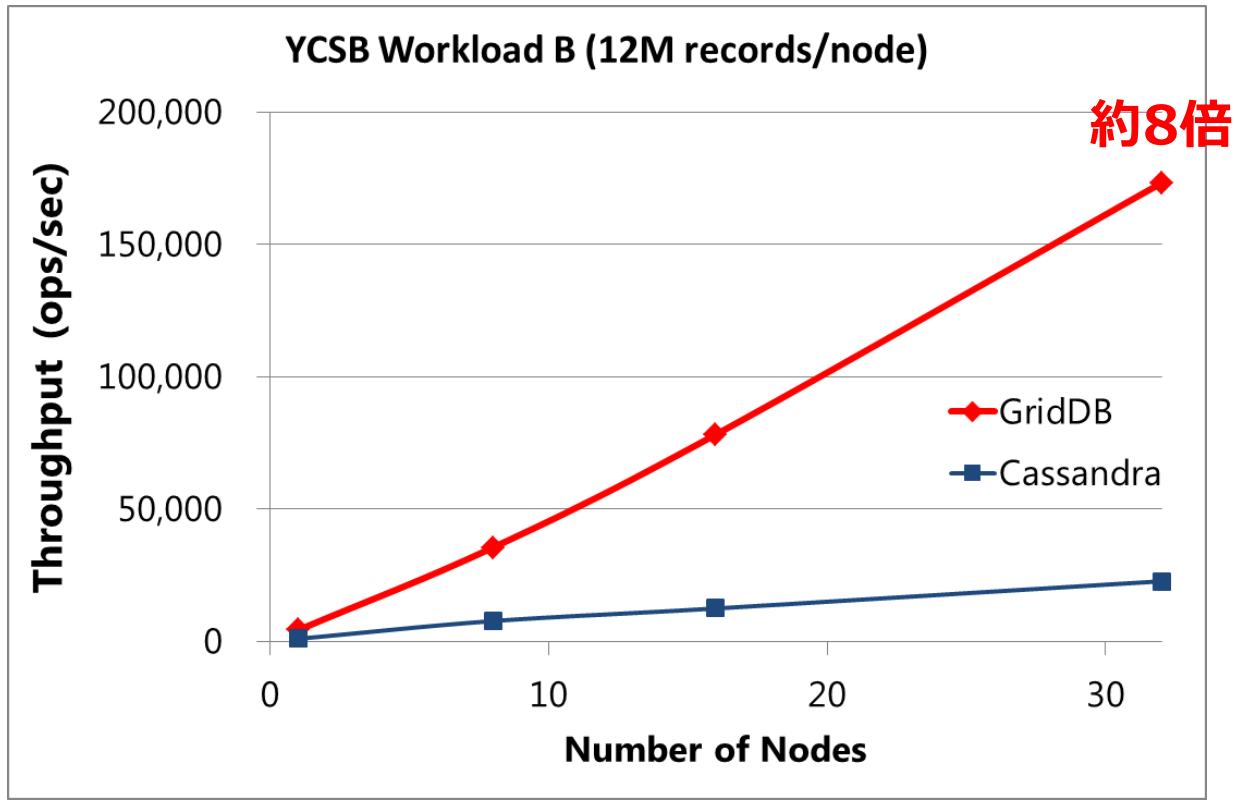


高性能

- **YCSB (Yahoo! Cloud Serving Benchmark)**
  NoSQLの代表的なベンチマーク　https://github.com/brianfrankcooper/YCSB



Read 50% + Write 50%



Read 95% + Write 5%

※フィックスターズ社によるYCSBベンチマーク結果

適用事例

- **社会インフラを中心に、高い信頼性・可用性が求められるシステムに適用中**

・フランス リヨン 太陽光発電 監視・診断システム
　　発電量の遠隔監視、発電パネルの性能劣化を診断
・クラウドBEMS
　　ビルに設置された各種メータの情報の収集、蓄積、分析
・石巻スマートコミュニティ プロジェクト
　　地域全体のエネルギーのメータ情報の収集、蓄積、分析
・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
・神戸製鋼所 産業用コンプレッサ稼働監視システム
　　グローバルに販売した産業用コンプレッサをクラウドを利用して稼働監視
・東芝機械 IoTプラットフォーム
　　工作機器、射出成形、ダイカストマシン、など膨大な製造データを管理
・デンソー Factory-IoT (IoTを活用したダントツ工場)
　　https://www.youtube.com/watch?v=9-yf-XN1Bgg&feature=youtu.be (ビデオ)
・DENSO International Americaの次世代の車両管理システム
　　https://griddb.net/ja/blog/griddb-automotive/
・クラウド型IoTソリューション
….

- **東芝IoTアーキテクチャー「SPINEX」の構成ソリューション**

GridDBの使い方

クイックスタート(1/2)

「GridDB V4.0 CEのRPMによるインストール、1台構成でサンプル実行の例」

- **環境**
  - OS: CentOS 7.5　※デモはWin7のノートPC上にVirtualBoxでVMを立てて、ゲストOSをCentOS 7.5とした。

1. **インストール**

```
$ wget https://github.com/griddb/griddb_nosql/releases/download/v4.0.0/griddb_nosql-4.0.0-1.linux.x86_64.rpm
$ rpm –ivh griddb_nosql-4.0.0-1.linux.x86_64.rpm　※OS上にユーザ”gsadm”が生成される
```

2. **設定**
   - ユーザ”gsadm”のパスワード設定

     ```
     $ passwd gsadm
     ```

   - ユーザ”gsadm”でログイン　※環境変数 $GS_HOME, $GS_LOGが設定される

     ```
     $ su – gsadm
     ```

   - GridDB管理ユーザ”admin”のパスワード設定

     ```
     $ gs_passwd admin
     ```

   - クラスタ名の設定

     ```
     $ vi conf/gs_cluster.json
     ```

     ※”clusterName”:””の部分にクラスタ名を入力する

クイックスタート(2/2)

3. ノードの起動、クラスタ構成

$ gs_startnode

$ gs_joincluster -c (クラスタ名) -u admin/(パスワード) –s (接続先IPアドレス)

4. サンプルプログラムの実行

$ export CLASSPATH=${CLASSPATH}:/usr/share/java/gridstore.jar

$ mkdir gsSample

$ cp /usr/griddb-X.X.X/docs/sample/program/Sample1.java gsSample/.

$ javac gsSample/Sample1.java

$ java gsSample/Sample1 239.0.0.1 31999 (クラスタ名) admin (パスワード)

→ Person: name=name02 status=false count=2 lob=[65, 66, 67, 68, 69, 70, 71, 72, 73, 74]

※ 239,0,0,1 (31999)はマルチキャストで通信するためのIPアドレス (ポートNo)

5. クラスタ停止、ノード停止

$ gs_stopcluster –u admin/(パスワード) –s (接続先IPアドレス)

$ gs_stopnode –u admin/(パスワード) –s (接続先IPアドレス)

ディレクトリ構成

- **基本ディレクトリ**
  - 実行ファイル
    - サーバモジュール (bin/gsserver)
    - 運用コマンド (bin/gs_startnode, gs_joincluster, …)
  - コンフィグファイル
    - ノード定義ファイル (conf/gs_node.json)
    - クラスタ定義ファイル (conf/gs_cluster.json)
  - その他
    - Javaクライアントライブラリ (gridstore.jar)
- **データディレクトリ**
  - データベースファイル
    - チェックポイントファイル (data/gs_cp_*.dat)
    - DB更新ログファイル (data/gs_log_*.log)
  - イベントログファイル (log/gridstore_*.log)

(インストールディレクトリ)
/usr/griddb-X.X.X/
    3rd_party/
    bin/
    conf/
    docs/

(gsadmのホームディレクトリ)
/var/lib/gridstore/
    conf/
    data/
    log/

コンフィグファイルの設定例（抜粋）

gs_node.json

```
{
   "dataStore":{
      "dbPath":"data",
      "syncTempPath":"sync",
      "storeMemoryLimit":"1024MB",
      "storeWarmStart":false,
      "concurrency":4,
      "logWriteMode":1,
      "persistencyMode":"NORMAL",
      "affinityGroupSize":4
   },
   "checkpoint":{
      "checkpointInterval":"60s",
      "checkpointMemoryLimit":"1024MB",
      "useParallelMode":false
   },
   "cluster":{
      "servicePort":10010
   },
   "sync":{
      "servicePort":10020
   },
   "system":{
      "servicePort":10040,
```

gs_cluster.json

```
{
   "dataStore":{
      "partitionNum":128,
      "storeBlockSize":“64KB"
   },
   "cluster":{
      "clusterName":“osc123",
      "replicationNum":2,
      "notificationAddress":"239.0.0.1",
      "notificationPort":20000,
      "notificationInterval":"5s",
      "heartbeatInterval":"5s",
      "loadbalanceCheckInterval":"180s"
   },
   "sync":{
      "timeoutInterval":"30s"
   },
   "transaction":{
      "notificationAddress":"239.0.0.1",
      "notificationPort":31999,
      "notificationInterval":"5s",
      "replicationMode":0,
      "replicationTimeoutInterval":"10s"
   }
```

運用コマンドの操作手順

1. **起動(各ノードについて。起動するマシン上で実行)**
   - gs_startnode
2. **クラスタ構成(各ノードについて。１台構成でも必要)**
   - gs_joincluster [-s 接続先IPアドレス:ポート] -n 構成ノード数 -c クラスタ名 -u ユーザ名/パスワード
3. **アプリ実行**
4. **クラスタ停止(クラスタを構成している1ノードに対して)**
   - gs_stopcluster [-s 接続先IPアドレス:ポート] -u ユーザ名/パスワード
5. **ノード停止(各ノードについて)**
   - gs_stopnode [-w [WAIT_TIME]][-s 接続先IPアドレス:ポート] [-f] -u ユーザ名/パスワード

※STAT取得(各ノードについて)

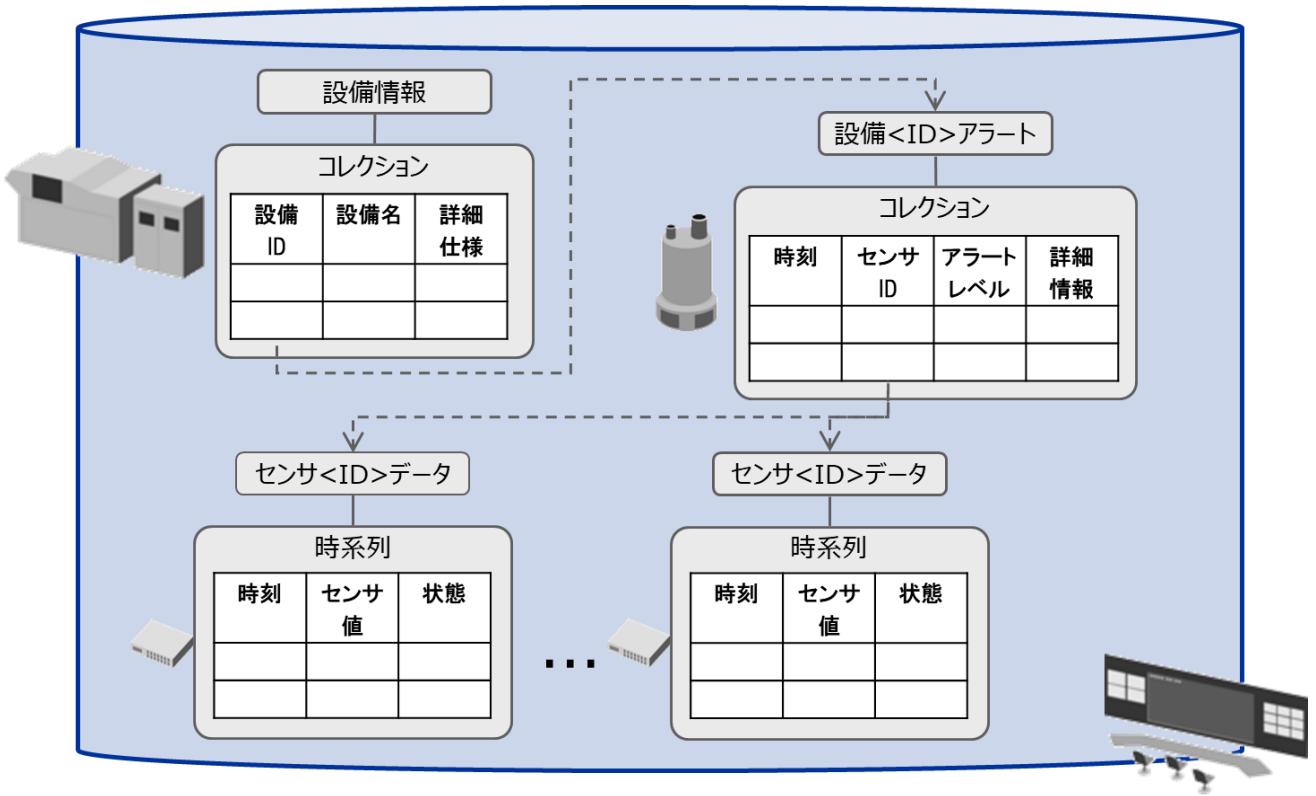
- gs_stat [-s 接続先IPアドレス:ポート] -u ユーザ名/パスワード

gs_stat出力例（抜粋）

```
$ gs_stat -u admin/admin
{
"cluster": {
        "activeCount": 1,
        "clusterName": “test",
        "clusterStatus": "MASTER",
        "designatedCount": 1,
        "master": {
            "address": "127.0.0.1",
            "port": 10040
        },
        "nodeList": [
            {
                "address": "127.0.0.1",
                "port": 10040
            }
        ],
        "nodeStatus": "ACTIVE",
        "notificationMode": "MULTICAST",
        "partitionStatus": "NORMAL",
        "startupTime": "2018-10-10T12:04:45+0900",
    },
    "currentTime": "2018-10-10T12:05:23+0900",
"performance": {
        "checkpointFileSize": 262144,
        "checkpointMemory": 0,
        "checkpointMemoryLimit": 1073741824,
        "peakProcessMemory": 67137536,
        "processMemory": 67137536,
        …
        "storeMemory": 0,
        "storeMemoryLimit": 1073741824,
        "storeTotalUse": 0,
        "swapRead": 0,
        "swapReadSize": 0,
        "swapReadTime": 0,
        "swapWrite": 0,
        "swapWriteSize": 0,
        "swapWriteTime": 0,
        "totalReadOperation": 0,
        "totalRowRead": 0,
        "totalRowWrite": 0,
        "totalWriteOperation": 0
    },
    "version": "4.0.0-33128 CE"
}
```

クライアント操作の例

1. プロパティ設定（主に接続用）
2. コンテナの生成・取得
3. (A) データ登録
   (B) 検索



クライアント操作の例：プロパティ設定

1. **プロパティ設定（主に接続用）**
2. コンテナの生成・取得
3. (A) データ登録
   (B) 検索

マルチキャスト方式の場合：

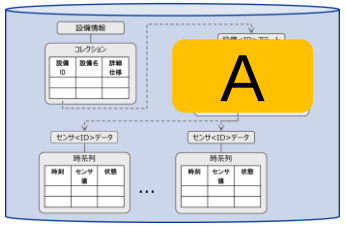
マルチキャストで通信するためのIPアドレス、ポートNoを指定する。

```
// プロパティ設定
Properties props = new Properties();
props.setProperty(“notificationAddress”, args[0]); // マルチキャスト・アドレス
props.setProperty(“notificationPort”, args[1]);// マルチキャスト・ポートNo
props.setProperty(“clusterName”, args[2]); // クラスタ名
props.setProperty(“user”, args[3]); // ユーザ名
props.setProperty(“password”, args[4]); // パスワード

// GridStoreインスタンスの取得
GridStore store = GridStoreFactory.getInstance().getGridStore(props);
```

クライアント操作の例：コンテナ生成、データ登録

1. プロパティ設定（主に接続用）
2. **コンテナの生成・取得**
3. **(A) データ登録**
   (B) 検索

```
// アラート情報
class Alert {
    @RowKey Long id;
    Date   timestamp;
    String sensorId;
    int    level;
    String detail;
}
// ロウキーには@RowKeyを付ける
```

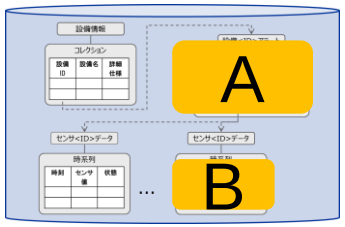



```
// コレクションコンテナ生成
Collection<Long,Alert> alertCol
    = store.putCollection(alertColName, Alert.class);

// 索引設定
alertCol.createIndex("timestamp");
alertCol.createIndex("level");
alertCol.setAutoCommit(false);
…
Alert alert = new Alert();
while ((nextLine = reader.readNext()) != null) {
    …
    alert.timestamp  = new Date(datetime);
    alert.sensorId   = nextLine[2];
    alert.level      = Integer.valueOf(nextLine[3]);
    alert.detail     = nextLine[4];

    // データ登録
    alertCol.put(alert);
}
alertCol.commit();
```

クライアント操作の例：検索

1. プロパティ設定（主に接続用）
2. **コンテナの**生成**・取得**
3. (A) データ登録
   (B) 検索

24時間以内に発生した重大アラートについて直前の時系列データを検索




```
Collection<String,Alert> alertCol
    = store.getCollection(alertColName, Alert.class); // コンテナ取得
…
// 24時間以内の重大アラート(level>3)を検索
String from = TimeStampUtil.format( TimeStampUtil.add(new Date(now), -24, TimeUnit.HOUR) );
Query<Alert> alertQuery
    = alertCol.query("select * where level > 3 and time > TIMESTAMP('" + from + "')" );// TQLによる検索

RowSet<Alert> alertRs = alertQuery.fetch();
while( alertRs.hasNext() ) {
    // 重大アラート発生したセンサのデータが入っている時系列コンテナを取得
    Alert alert = alertRs.next();
    TimeSeries<Point> ts = store.getTimeSeries(alert.sensorId, Point.class); // 時系列コンテナ取得

    // 直前の時系列データを検索
    Date endDate   = alert.timestamp;
    Date startDate = TimeStampUtil.add(alert.timestamp, -10, TimeUnit.MINUTE);
    RowSet<Point> rowSet =  ts.query(startDate, endDate).fetch();  // 専用関数による範囲検索

    while (rowSet.hasNext()) {
        Point ret = rowSet.next();
```

※詳細は以下をご覧ください。
GridDB プログラミングチュートリアル：https://griddb.net/ja/docs/manuals/v3.1/GridDB_ProgrammingTutorial.html
ソース：https://github.com/griddb/griddb_nosql/tree/master/sample/tutorial/ja

はまりやすいポイント

1. 起動できない

- 環境変数が未設定エラー。⇒ gsadmでログインしていない。
  - gsadmのパスワードを設定した上でgsadmでログイン。環境変数が自動設定される。
  - suコマンドでログインする場合、「$ su – gsadm」のように「-」/「-l」オプションを付けること。
- 管理ユーザ（例：admin）のパスワードが設定されていない。
- gs_cluster.jsonファイルにクラスタ名が設定されていない。
- 「$ hostname -i」にてホスト名からIPアドレスが取得できない。
  ⇒ /etc/hostsファイルの設定を確認する。

2. 運用コマンド操作ができない

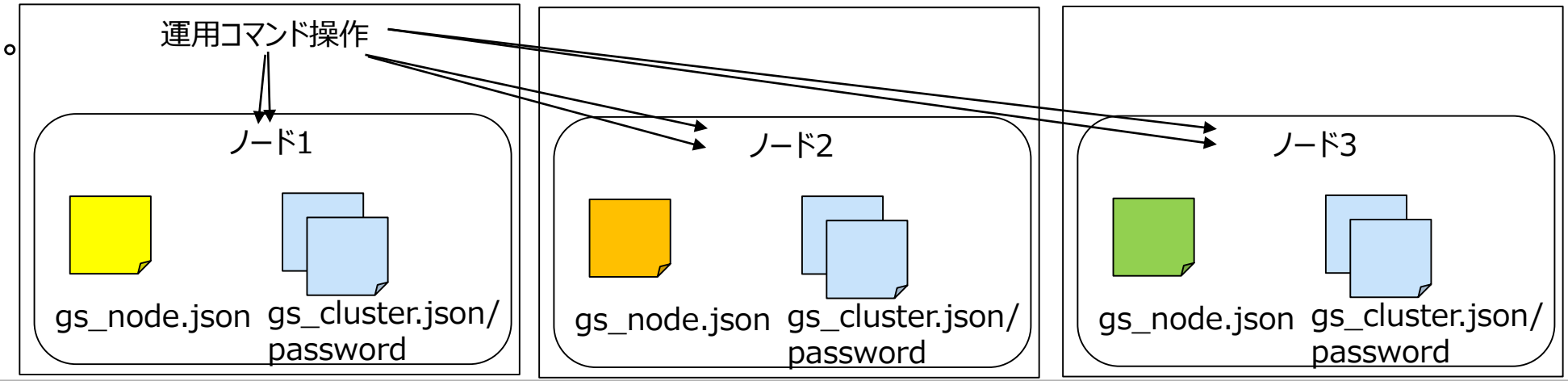
- プロキス変数が設定されている。⇒プロキシ側にアクセスしないようにする。
  「(例)$ export no_proxy=10.0.2.15」

3. クライアント操作ができない

- ファイアウォールに通信用ポートNoを許可する。
  「(例)$ firewall-cmd --zone=public --add-port=31999/udp」

N台でクラスタを構成する場合

- **コンフィグ**
  - gs_cluster.json, passwdファイルはクラスタ内の全ノードで共通の設定にする。
- **運用コマンド操作**
  - N台の各ノードについて、起動およびクラスタ構成などの操作を行う。
    - 起動：リモートマシン操作ではsshを使う。
    - クラスタ構成：クラスタ構成数を指定する (例：-n 3)。
  - クラスタ停止はクラスタを構成している1ノードについて操作を行えばよい。
- **クライアント操作**
  - 1台構成と同様。



AWS/Azureを使う場合

- **マルチキャスト方式以外の方法を使う**

固定リスト方式の場合：全ノードのIPアドレス、通信用ポートNoを列挙する。

サーバ側の設定 (gs_cluster.json)

```
"cluster":{
    "notificationMember": [
        { "cluster": {"address":"172.17.0.44", "port":10010}, "sync": {"address":"172.17.0.44", "port":10020},
          "system": {"address":"172.17.0.44", "port":10040}, "transaction": {"address":"172.17.0.44", "port":10001} },
        { "cluster": {"address":"172.17.0.45", "port":10010}, "sync": {"address":"172.17.0.45", "port":10020},
          "system": {"address":"172.17.0.45", "port":10040}, "transaction": {"address":"172.17.0.45", "port":10001} },
        { "cluster": {"address":"172.17.0.46", "port":10010}, "sync": {"address":"172.17.0.46", "port":10020},
          "system": {"address":"172.17.0.46", "port":10040}, "transaction": {"address":"172.17.0.46", "port":10001} } ]
```

クライアント側の設定 (Javaプログラム)

```
// プロパティ設定
Properties props = new Properties();
props.setProperty("notificationMember", "172.17.0.44:10001, 172.17.0.45:10001, 172.17.0.46:10001");
    // (アドレス1):(ポート1),(アドレス2):(ポート2),...形式
…
GridStore store = GridStoreFactory.getInstance().getGridStore(props);
```

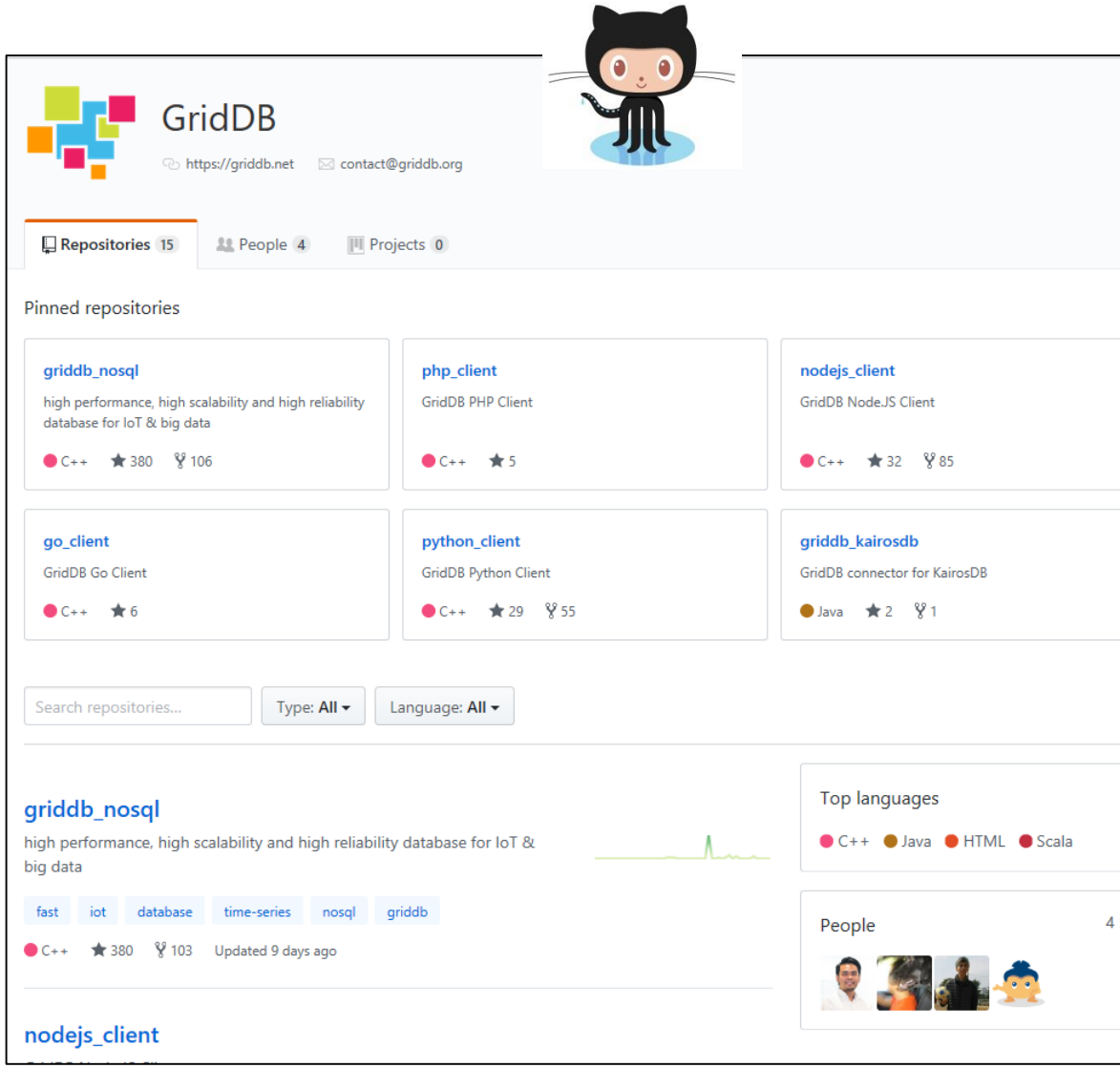
OSS活動

GitHubサイト　https://github.com/griddb/

griddb github | 検索

- **NoSQL機能、様々な開発言語のクライアント、主要OSSとのコネクタをソース公開**

  https://github.com/griddb/griddb_nosqlなど
- **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



最近のOSS活動

2018年

- 2月　GridDB Pythonクライアントのソース公開(GitHub)
- 3月　GridDB Goクライアントのソース公開(GitHub)
- 5月　GridDB V4.0 Community Editionのソース公開(サーバ、Javaクライアント) (GitHub)
- 6月　GridDB Node.JSクライアントのソース公開(GitHub)
  GridDB PHPクライアントのソース公開(GitHub)
- 7月　GridDB V4.0 CE Cクライアントのソース公開(GitHub)
- 8月　GridDB用Kafkaコネクタ･サンプルのソース公開(GitHub)
- 9月　GridDB Pythonクライアントのパッケージ公開(PyPI)
  GridDB Node.JSクライアントのパッケージ公開(npm)
- 10月　GridDB Perlクライアントのソース公開(GitHub)

GridDB V4.0 CE 追加機能

1. 大規模データ管理強化

– より少ないノード台数で柔軟に効率よく対応するための強化・改善を行いました。

2. 検索結果取得時の分割処理への対応

– バッファのサイズなどに収まるよう、内部で自動的に分割し処理することで、大量の検索結果が取得可能になりました。

3. コンテナ名などの名前に使用できる文字の拡張

– 他のデータベースとの連携を容易にするためにコンテナ名などに使用できる文字を拡張しました。

4. データベースファイルのサイズ縮小機能

– データベースファイルのサイズ(実ディスク割当量)を縮小することが可能になりました。

5. Cクライアントのライセンス変更

– AGPL V3からApache V2に変更しました。

公開状況

性能測定 | 収集 | 可視化 | 分散処理 | 分析 | Webアプリ

③GitHub以外のサイトからの情報発信

PyPI (https://pypi.org/)

npm (https://www.npmjs.com/)

①主要OSSとの連携強化

YCSB コネクタ | Kafka コネクタ | KairosDB コネクタ | Hadoop MapReduce コネクタ | Spark コネクタ

Javaクライアント

②各種クライアントの拡充

Python クライアント | Node.JS クライアント | Go クライアント | PHP クライアント | Ruby クライアント

Cクライアント

GridDB V4.0 CE(Community Edition)

GitHub (https://github.com/)

④主要OSSリポジトリへのコントリビュート（YCSBなど予定）

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
  - マニュアル
  - サンプルコード

  など
- **コミュニケーションの場(フォーラム)を提供**

最近のブログ

09-29-2018

GridDB PHPクライアント入門

わたしたちは、GridDBのPHPプログラミング言語用のデータベースコネクタをリリー
このコネクタはPHP
ジョンをサポートし
CentOSバージョン
します。

09-29-2018

Fixed ListでGridDBを使う方法

FIXED_LISTモードとMulticastモードでGridDBを実行する場合、どのような違いがあるので

09-29-2018

GridDB Node.jsクライアントを使ってみよう

GridDBは近年非常に人気のあるnode.jsとのデータベースコネクタ

09-29-2018

PostgreSQL用のGridDB外部データラッパーを使ってみよう

複数の異なるインタフェースから

06-09-2018

GridDBのGo言語 Client入門

GridDBは、Googleのプログラミング言語Go用のデータベースコ

06-06-2018

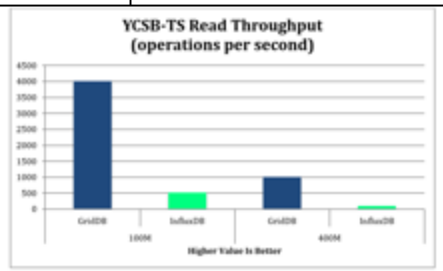
GridDB Community Edition バージョン4.0

GridDB Community Editionがアップデートされました。バージョン4.0では、コンテナ名やクラスター名の文字の幅を広げるなど、

04-05-2018

PostgreSQLからGridDBへの移行

このブログでは、PostgreSQL databaseをPythonを使ってGridDBへ移行する方法をご紹介します。



04-05-2018

GridDBとInfluxDBのTimeSeriesベンチマーク比較

時系列(Timeseries) データベースであるGridDBとInfluxDBを、CentOS 6.9 image を使った

ツイッター https://twitter.com/griddb_jp

griddb_jp | 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)**

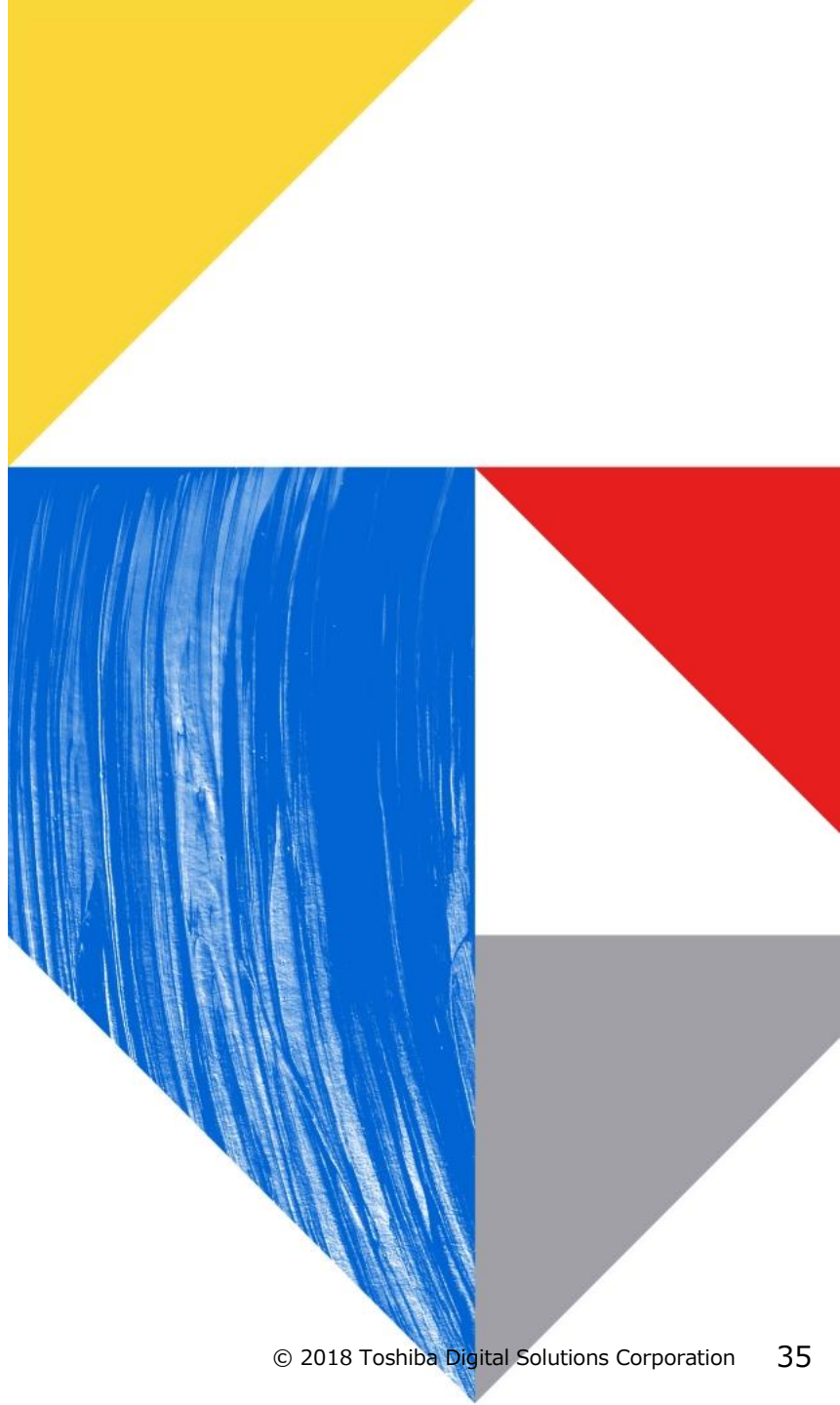
まとめ

- **GridDBはビッグデータ・IoT向けのスケールアウト型データベースです。**
- **今回、GridDBの概要、使い方についてご説明いたしました。**
- **OSSサイト、デベロッパーズサイトなどで、GridDB機能強化やクライアント拡充、主要OSSとの連携強化などの様々なOSS活動を実施しています。**

オープンソースのGridDBを是非とも使ってみてください。

TOSHIBA

ご清聴ありがとうございました



ご参考：GridDBに関する情報

- **GridDB デベロッパーズサイト**
  - https://griddb.net/
- **GridDB OSSサイト**
  - https://github.com/griddb/griddb_nosql/
- **Twitter: GridDB (日本)**
  - https://twitter.com/griddb_jp
- **Twitter: GridDB Community**
  - https://twitter.com/GridDBCommunity
- **Facebook: GridDB Community**
  - https://www.facebook.com/griddbcommunity/

- **GridDB お問い合わせ**
  - デベロッパーズサイトのフォーラム、OSSサイトのGitHubのIssue、
    もしくはcontact@griddb.orgをご利用ください

ご参考：デベロッパーズサイトの主なコンテンツ（ハウツーもの）

- **AWS/Azure上で動かす方法**
  - 「GridDB Azureクラスタの構築」
  - 「Fixed ListでGridDBを使う方法」
- **Docker上で動かす方法**
  - 「Docker上でGridDBを実行する」
- **Pappetによる設定方法**
  - 「Puppetを使用したGridDBの設定方法」
- **各種クライアント**
  - 「GridDB's C/Python/Ruby APIsを使ってみよう」
  - 「GridDBのGo言語 Client入門」
  - 「GridDB Node.jsクライアントを使ってみよう」
  - 「GridDBのPHPクライアント入門」
- **MapReduceコネクタ**
  - 「Hadoop MapReduce用のGridDBコネクタの使い方」
- **Sparkコネクタ**
  - 「Apache SparkのためのGridDBコネクタ」
- **YCSBコネクタ**
  - 「YCSB向けGridDBコネクタを使ってみよう」

※https://griddb.net/ja/blog/

ご参考：デベロッパーズサイトの主なコンテンツ(ホワイトペーパなど)

ホワイトペーパ：

- GridDB®とは
- GridDB と Cassandra のパフォーマンスとスケーラビリティ – Microsoft Azure 環境における YCSB パフォーマンス比較
- GridDBとInfluxDBのTimeSeriesベンチマーク比較
- GridDB Reliability and Robustness

など

ブログ：

- IoT産業におけるGridDB導入事例
- 自動車産業におけるGridDB導入事例
- 自律型データ配信アルゴルズム (ADDA)
- CAP 定理と GridDB
- Raspberry Piチュートリアル：KairosDBコネクタを介してGridDBに温度データを送信する

など

付録：クライアント操作

1. プロパティ設定（主に接続用）
2. コンテナ生成･取得
3. (A) データ登録･取得
   (B) 検索

1．プロパティ設定

マルチキャスト方式の場合：マルチキャストで通信するためのIPアドレス、ポートNoを指定する。

```
// プロパティ設定
Properties props = new Properties();
props.setProperty(“notificationAddress”, args[0]); // マルチキャスト・アドレス
props.setProperty(“notificationPort”, args[1]);// マルチキャスト・ポートNo
props.setProperty(“clusterName”, args[2]); // クラスタ名
props.setProperty(“user”, args[3]); // ユーザ名
props.setProperty(“password”, args[4]); // パスワード

// GridStoreインスタンスの取得
GridStore store = GridStoreFactory.getInstance().getGridStore(props);
```

2．コンテナ生成･取得（静的）

```
【時系列コンテナ】
// スキーマ定義
static class Point {
    @RowKey Date timestamp; // ロウキーには@RowKeyをつける
    boolean active;
    double voltage;
}

// 時系列コンテナの作成
TimeSeries<Point> ts = store.putTimeSeries("point01", Point.class);

// 時系列コンテナの取得
TimeSeries<Point> ts = store.getTimeSeries("point01");
```

2．コンテナ生成（動的）

```
【時系列コンテナ】
// コンテナ情報を作成
ContainerInfo info = new ContainerInfo();
info.setType(ContainerType.TIME_SERIES); // コンテナタイプを設定
info.setName(“point01”); // コンテナ名を設定
info.setColumnInfoList(Arrays.asList(
    new ColumnInfo(“timestamp", GSType.TIMESTAMP),
    new ColumnInfo(“active", GSType.BOOL),
    new ColumnInfo(“voltage", GSType.DOUBLE)); // カラム情報（カラム名、カラムタイプ）を設定
info.setRowKeyAssigned(true); // ロウキーの有無を設定。有だと先頭カラムがロウキーとなる

// 時系列コンテナの作成
Container<Date, Row> container = store.putContainer(null, info, false);

// 時系列コンテナの取得
Container<Date, Row> container = store.getContainer(“point01”);
```

3(A). データ登録・取得

【時系列コンテナ】

```
// Rowのデータを用意
Point point = new Point();
point.timestamp = TimestampUtils.current(); // 現在時刻を設定
point.active = false;
point.voltage = 100;

ts.put(point); // 登録

point = ts.get(point.timestamp); // 取得
```

3(B). 検索

```
【時系列コンテナ】
// (A) コンテナ内のRowを検索
// 停止中にもかかわらず電圧が基準値以上の箇所を検索
Query<Point> query = ts.query(“select * from point01 where not active and voltage > 50");
RowSet<Point> rs = query.fetch();

// 検索したRowの取得
while (rs.hasNext()) {
    Point hotPoint = rs.next();
    Date hot = hotPoint.timestamp;
    System.out.println("[Alert] " + TimestampUtils.format(hot) + " hot=" + hotPoint.voltage);
}

// (B) 平均値
// 基準値以上の箇所について10分前後の平均値を求める
Date start = TimestampUtils.add(hot, -10, TimeUnit.MINUTE);
Date end = TimestampUtils.add(hot, 10, TimeUnit.MINUTE);
AggregationResult avg = ts.query("select AVG(voltage) where timestamp > ” + start.timestamp + “ and
timestamp < “ + end.timestamp);
System.out.println(" avg=" + avg.getDouble());
```