TOSHIBA
Leading Innovation

高速処理と高信頼性を両立し、ペタバイト級の多種大量データを蓄積する、ビッグデータ/ＩｏＴ時代のデータベースとは？？

株式会社 東芝
野々村 克彦

TOSHIBA
Leading Innovation

高速処理と高信頼性を両立し、ペタバイト級の多種大量データを蓄積する、ビッグデータ/ＩｏＴ時代のデータベースとは？？

スケールアウト型DB GridDB

株式会社 東芝
野々村 克彦

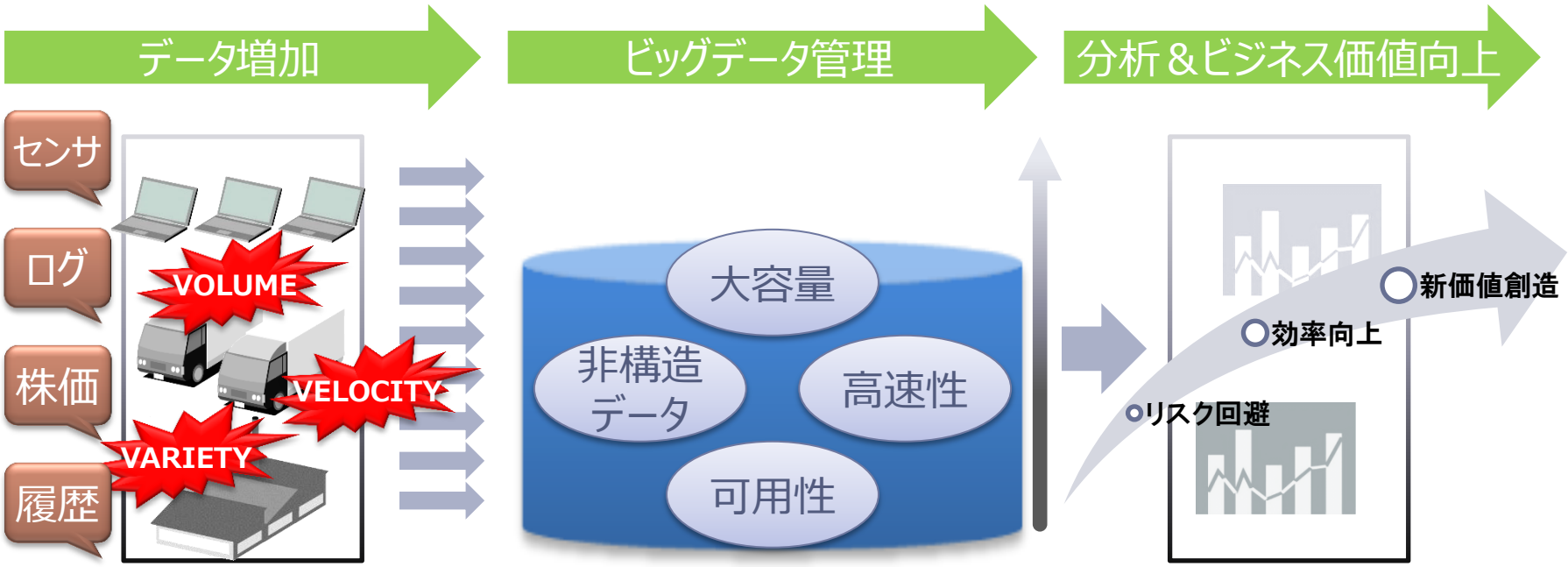
プロフィール

- **名前：野々村　克彦**
- **経歴：**
  - ００年代　XMLデータベースTX1　開発メンバ
  - １０年代　スケールアウト型DB GridDB(旧GridStore)　開発メンバ
  - １５年～　GridDB オープンソース・チーム
- **昨年から始めたこと：**
  - 息子（７歳）とサッカー
  - GitHub

目次

ビッグデータ

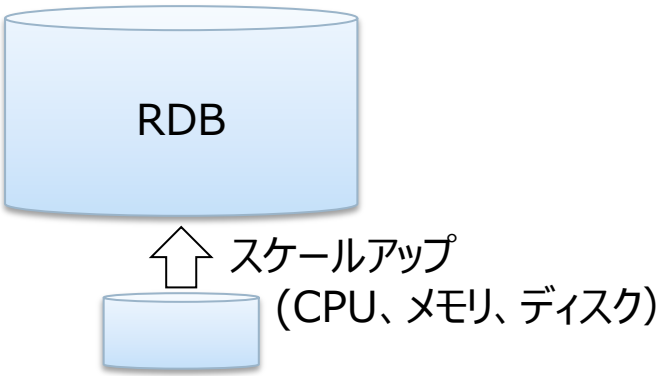
- **ビジネスの価値を向上させるビッグデータ活用が本格化**
  - センサーデータ、履歴データなど多様なデータが日々増加
- **ビッグデータ管理の要件に合わせたデータベースが必要**

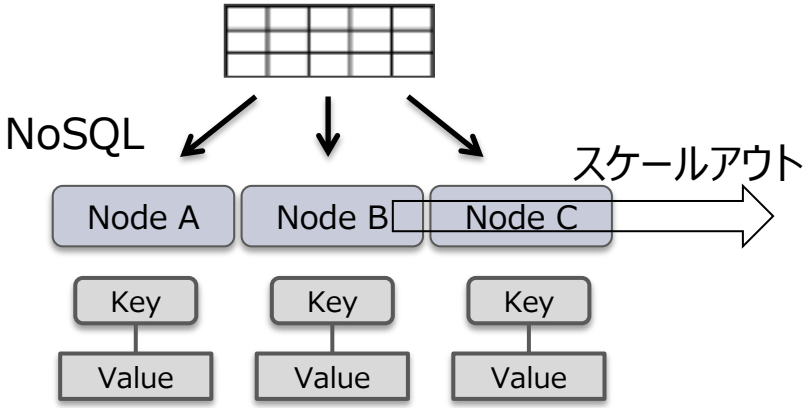


ビッグデータ管理は柔軟な拡張性が必須

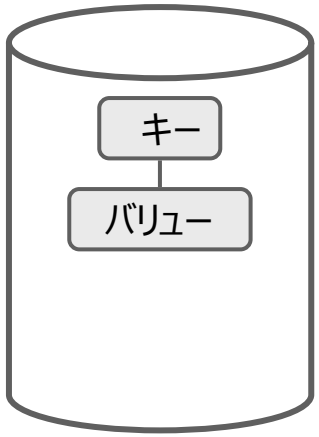
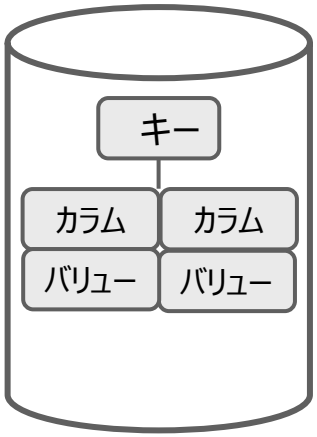
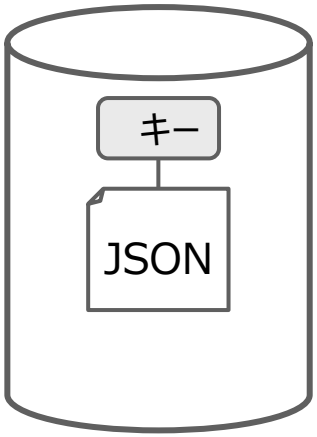
RDBとNoSQL

- **RDB**
  - スケールアップでは限界がある。ビッグデータを管理するのに適していない
  - 一貫性を重視するため、スケールアウトは困難である
- **NoSQL(Not Only SQL)**
  - キーによるPut/Getが基本I/F（キーバリュー型）
  - スケールアウトによる性能向上で近年注目されている
  - 一貫性を緩和する代わりにRDBでは対応できない規模の大容量データを管理可能である





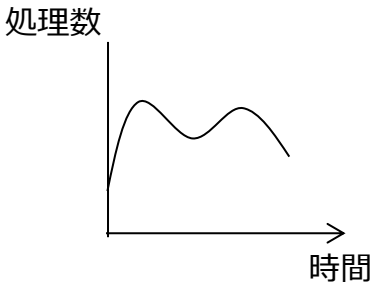
NoSQLのデータモデル

	キーバリュー型	列指向型	ドキュメント型
データモデル			
NoSQLの例	Riak	Cassandra	MongoDB

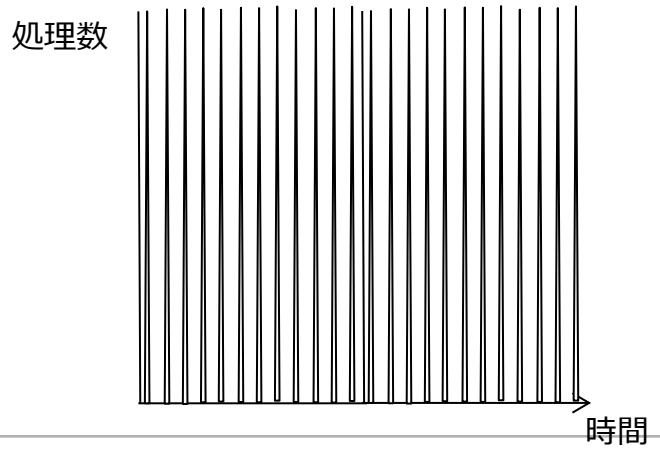
IoT

- **IoT(Internet of Things、モノのインターネット)**
  - 一意に識別可能な「もの」がインターネット/クラウドに接続され、情報交換することにより相互に制御する仕組み　※「IoT」『フリー百科事典　ウィキペディア日本語版』
- **特性**
  - 分、秒周期、さらにはそれ以下の周期で発生する膨大なセンサーデータを扱う必要がある。長時間に渡るデータを保持する必要がある。
  - 各センサ内のデータ欠損や参照データの矛盾など、データー貫性やデータ整合性を保つ必要がある。

人の活動で生成されるデータ
・SNS、ゲームなど
・テキスト、イメージデータ
・インメモリ前提



IoTデータ（センサー、ログ、履歴、株価等）

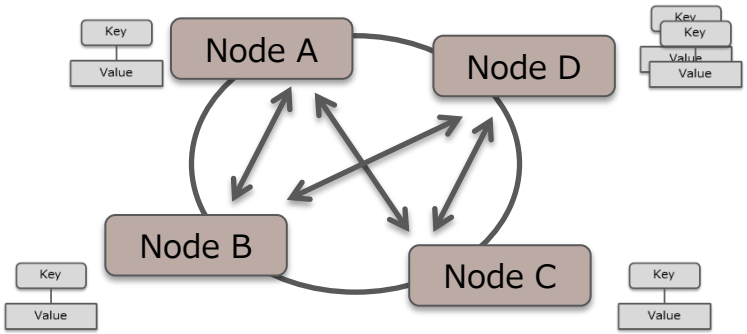
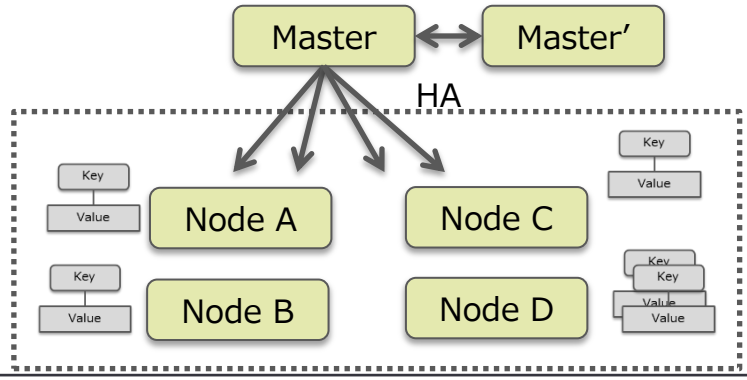


IoTにおける既存NoSQLの課題

（A）IoTのデータ管理が困難

– センサ単位の一貫性を保てない。時間範囲指定の検索ができない、
メモリが足りない場合に性能が大幅に劣化、など

（B）既存クラスタ管理方式に起因するトレードオフ問題

ピアツーピア(Peer to Peer)	マスタスレーブ(Master Slave)
	
○ノード追加でのデータ再配置が容易 ×一貫性維持のためのノード間通信のオーバヘッドが大⇒一貫性と処理速度がトレードオフ	○一貫性の維持は容易 ×マスタノードが単一障害点(SPOF) ×ノード追加でのデータ再配置が難しい

IoTにおける既存NoSQLの課題

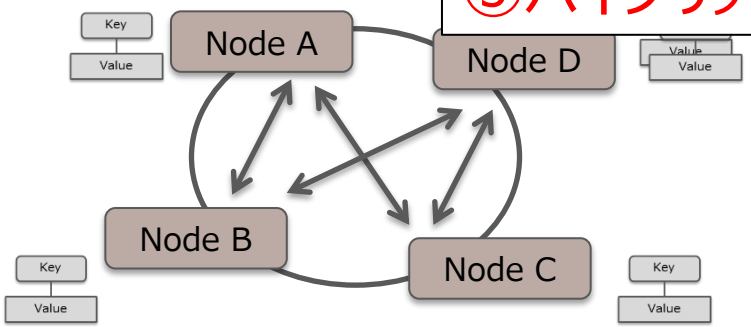
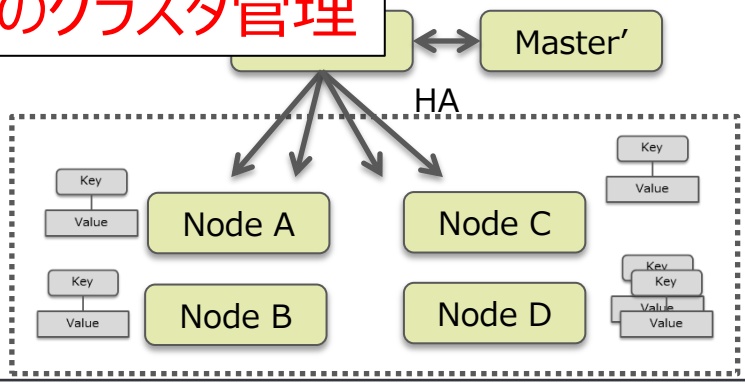
（A）IoTのデータ管理が困難

– センサ単位の一貫性を保てない、時間範囲指定の検索ができない、
メモリが足りない

①キーコンテナ型のデータモデル

（B）既存クラスタ管理方式に起因するトレードオフ問題

③ハイブリッド型のクラスタ管理

ピアツーピア(Peer to Peer)	マスタスレーブ(Master Slave)
	
○ノード追加でのデータ再配置が容易 ×一貫性維持のためのノード間通信のオーバヘッドが大⇒一貫性と処理速度がトレードオフ	○一貫性の維持は容易 ×マスタノードが単一障害点(SPOF) ×ノード追加でのデータ再配置が難しい

目次

GridDBのオープンソース化

- **GridDBとは**
  - IoTデータ管理向けのスケールアウト型DB
- **GitHub上にNoSQL機能をソース公開(2016/2/25)**
  - (英語)https://github.com/griddb/griddb_nosql/
  - (日本語)https://github.com/griddb/griddb_nosql/README_ja.md

- **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

GridDBの特長

① **IoT向けデータモデル**
- キーコンテナ型のデータモデル

② **高パフォーマンス(High Performance)**
- メモリ指向アーキテクチャ

③ **高信頼性(High Reliability)**
- （P2Pとマスタスレーブの）ハイブリッド型のクラスタ管理技術

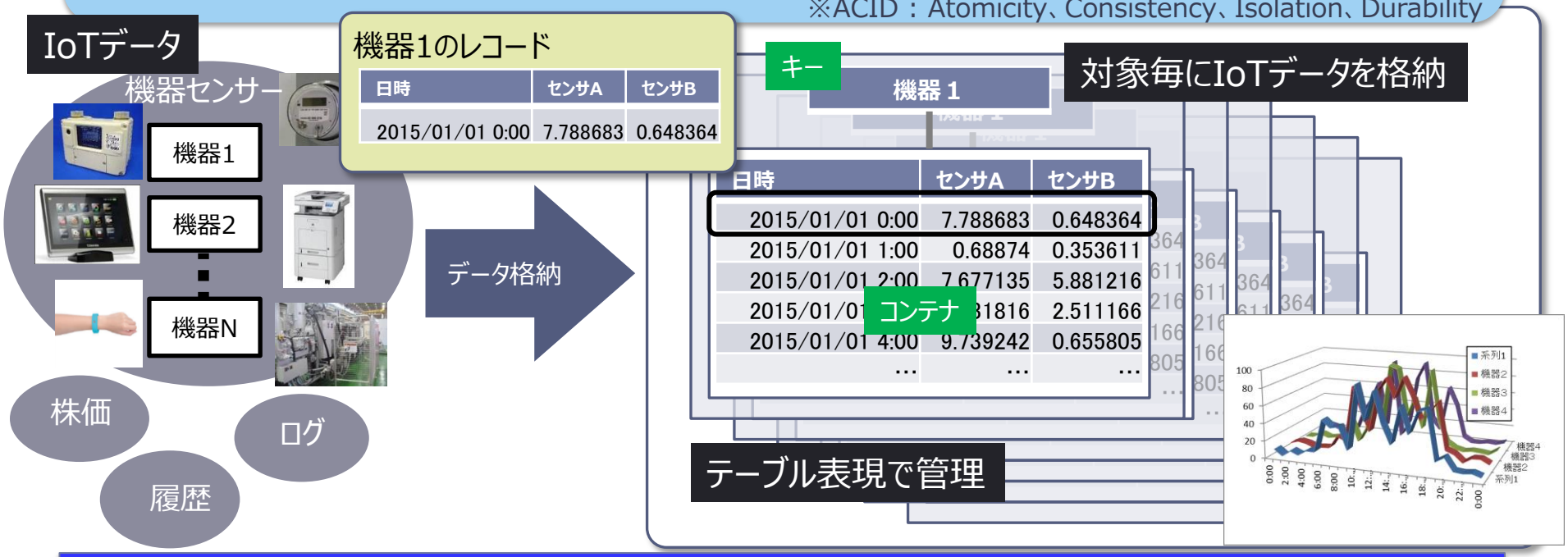
④ **高スケーラビリティ(High Scalability)**
- 自律データ再配置（ＡＤＤＡ）技術

① IoT向けのデータモデル

キーコンテナ型のデータモデル

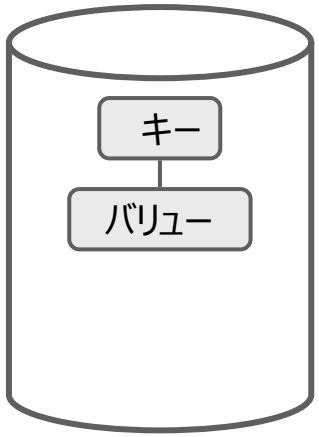
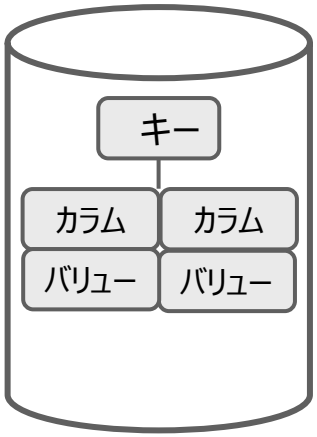
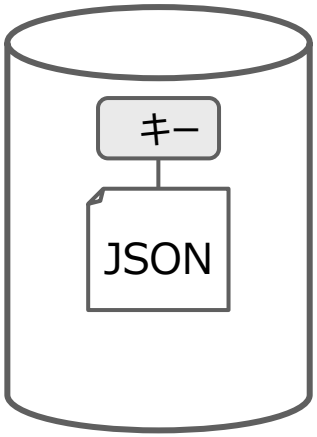
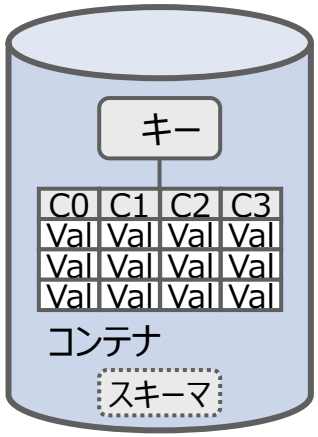
- キーバリューをグループ化するコンテナ（テーブル）
- コンテナのスキーマ定義が可能。カラムにインデックスを設定可能
  SQLライクなクエリ(TQL)が利用可能
- レコード単位でトランザクション操作（コンテナ単位でACID保証）

※ACID：Atomicity、Consistency、Isolation、Durability



単純なキーバリュー型とは異なり、使い慣れたRDBに近いモデリングが可能

データモデルの比較

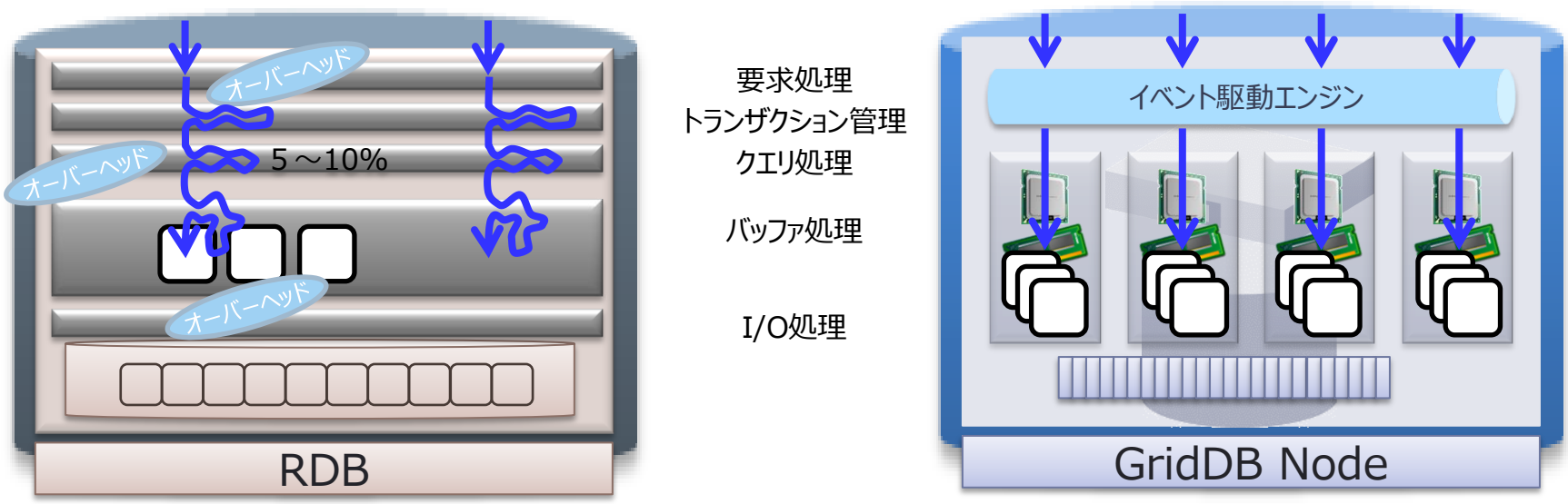
	キーバリュー型	列指向型	ドキュメント型	**キーコンテナ型**
データモデル				
NoSQLの例	Riak	Cassandra	MongoDB	**GridDB**

- **コンテナの種類**
  - コレクションコンテナ：レコード管理用
  - 時系列コンテナ：時刻で並べられたレコード集合。時系列データ管理用
    - 期限解放機能、サンプリング機能など

② 高パフォーマンス

メモリ指向アーキテクチャ

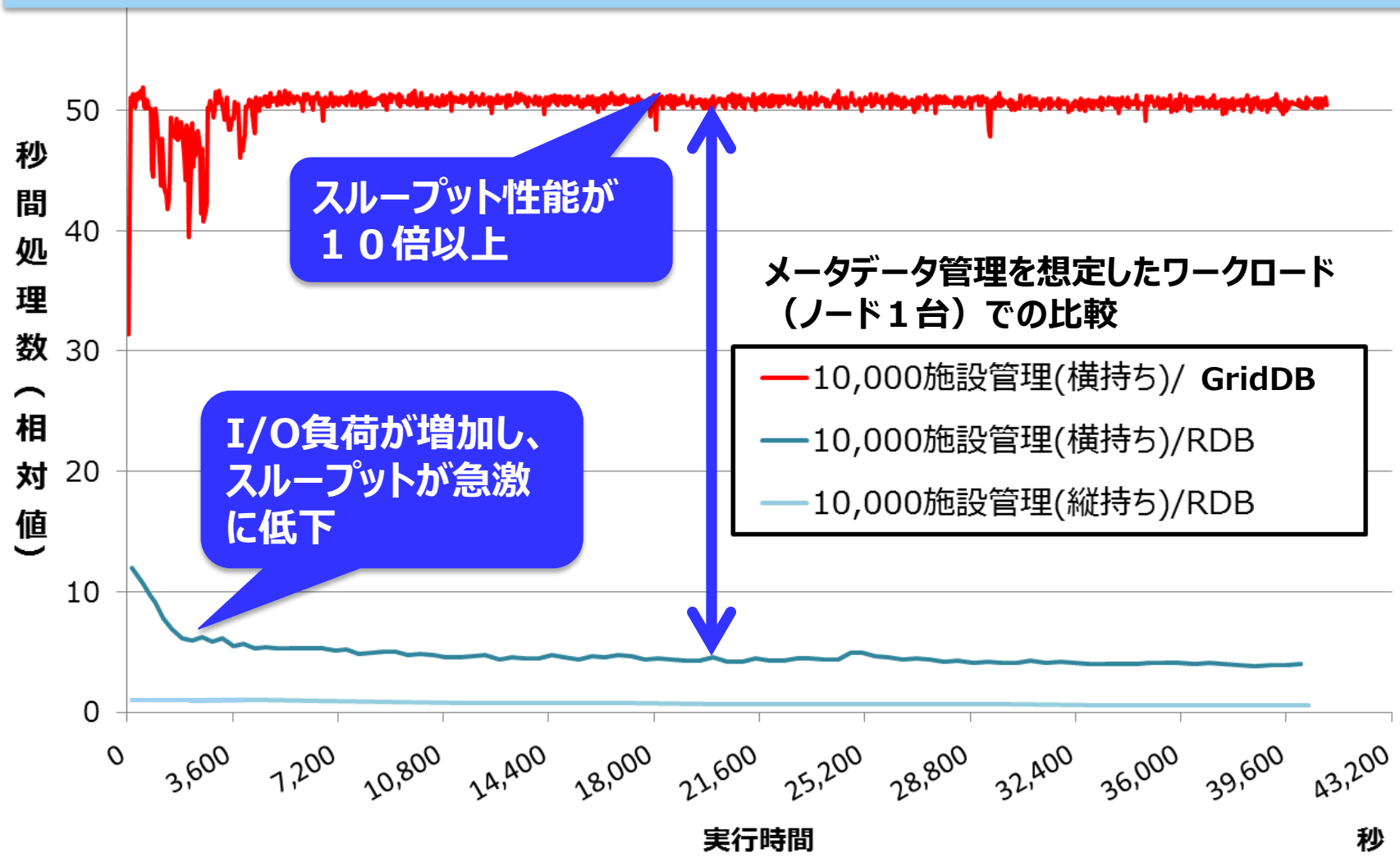
- イベント駆動エンジンであるため、少ないリソースで多くの要求を無駄なく処理
- メモリ、ディスクアクセスの排他処理や同期待ちを無くした、オーバヘッドの無いデータ処理
- GB超級のメモリ搭載を前提とし、読み書きサイズを最適化しI/O効率を改善



H/Wのスペックを最大限に生かすインメモリ指向DB

RDBとの性能比較

登録件数が１０億件超の規模になっても、安定した性能を維持



※RDBの10,000施設処理ケースでの最大値を1とした場合の相対値
横持ち：1時刻で機器毎のメータをまとめて（50メータで）1レコード作成
縦持ち：1時刻で1メータごとに1レコード作成

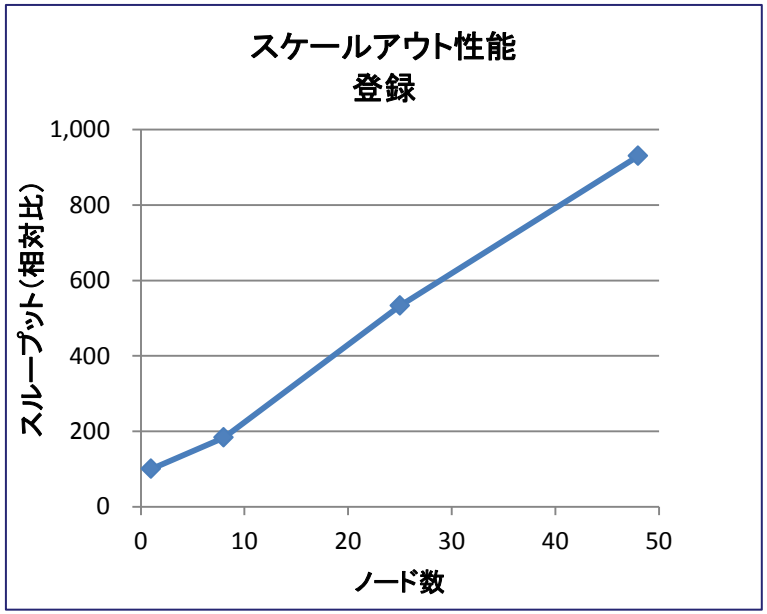
※1施設当たり50メータ

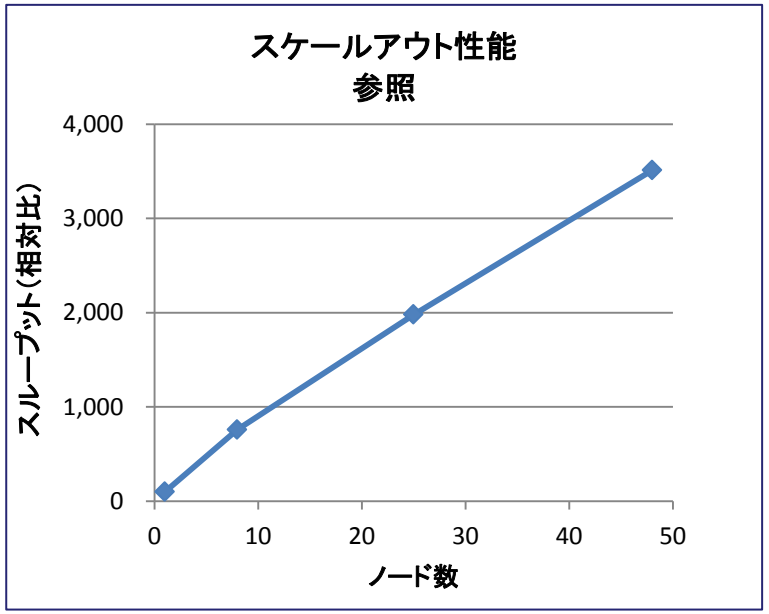
スケールアウト性能

- Yahoo Cloud Serving Benchmark （YCSB）による測定
- ノード数に対して
  性能がリニアに向上

http://labs.yahoo.com/news/yahoo-cloud-serving-benchmark

項目	値
台数	1～50台
ロウサイズ（レコードサイズ）	724B
件数	7.5億件
レプリカ数	3

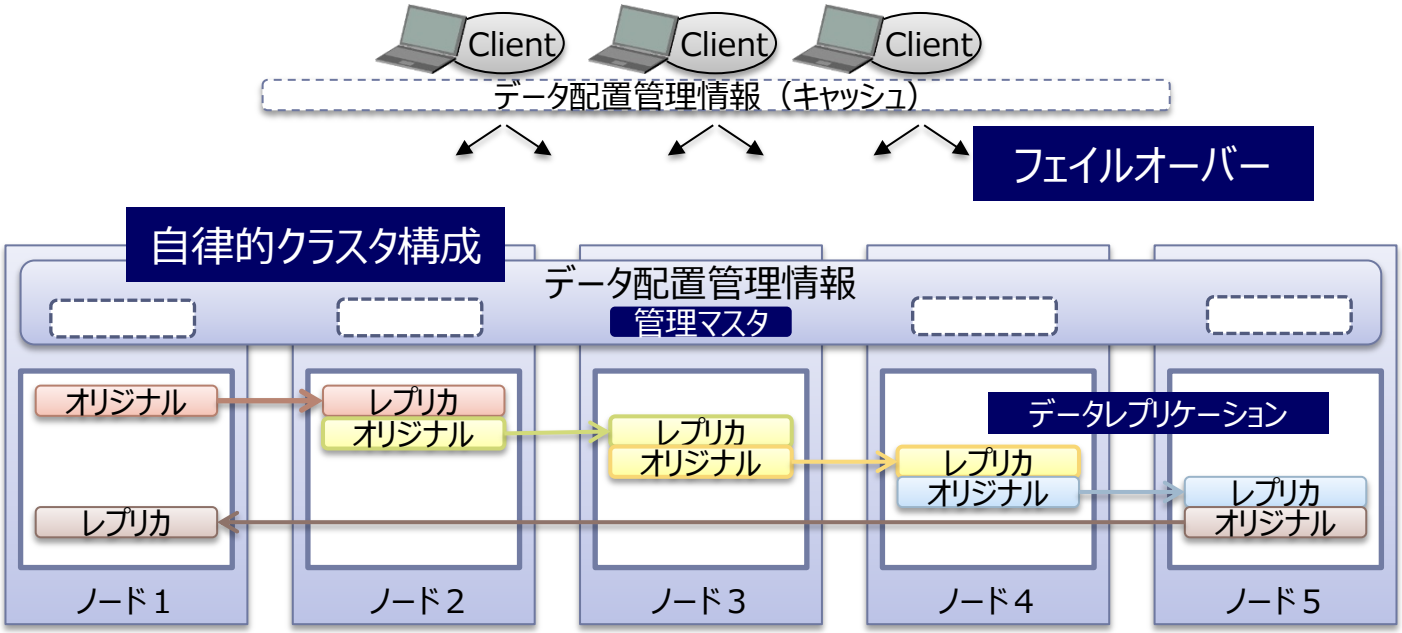




③ 高信頼性

ハイブリッド型クラスタ技術

- ノード間で自律的、動的にマスタノードを決定。単一故障点（ＳＰＯＦ）を排除
- レプリケーションによるデータ多重化、フェールオーバーが可能
- 永続化（インメモリ／ディスク併用）

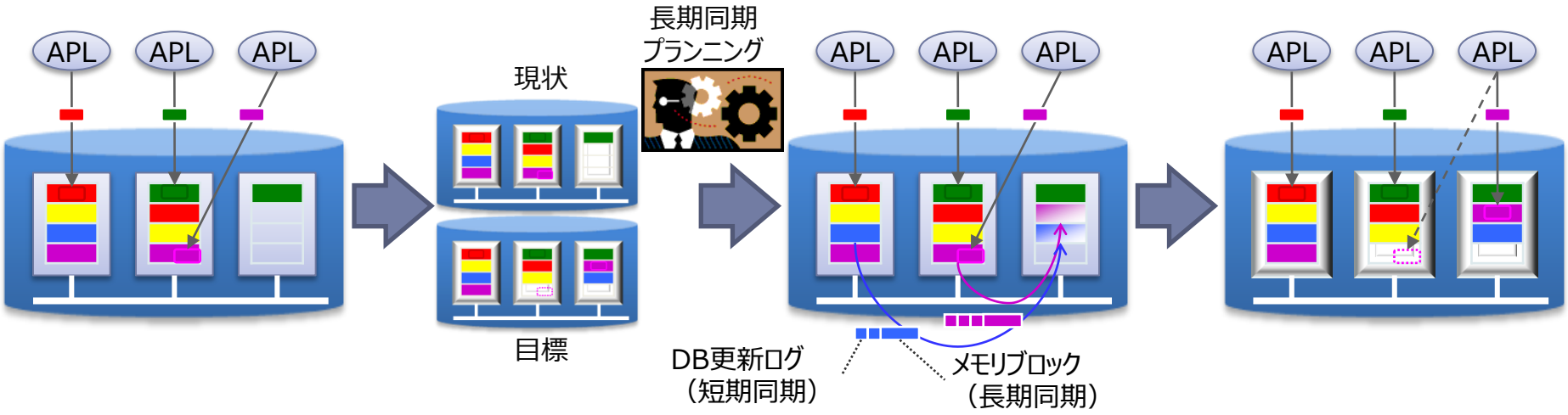


特別なスキルを必要とせずに、高可用なクラスタ構成が可能

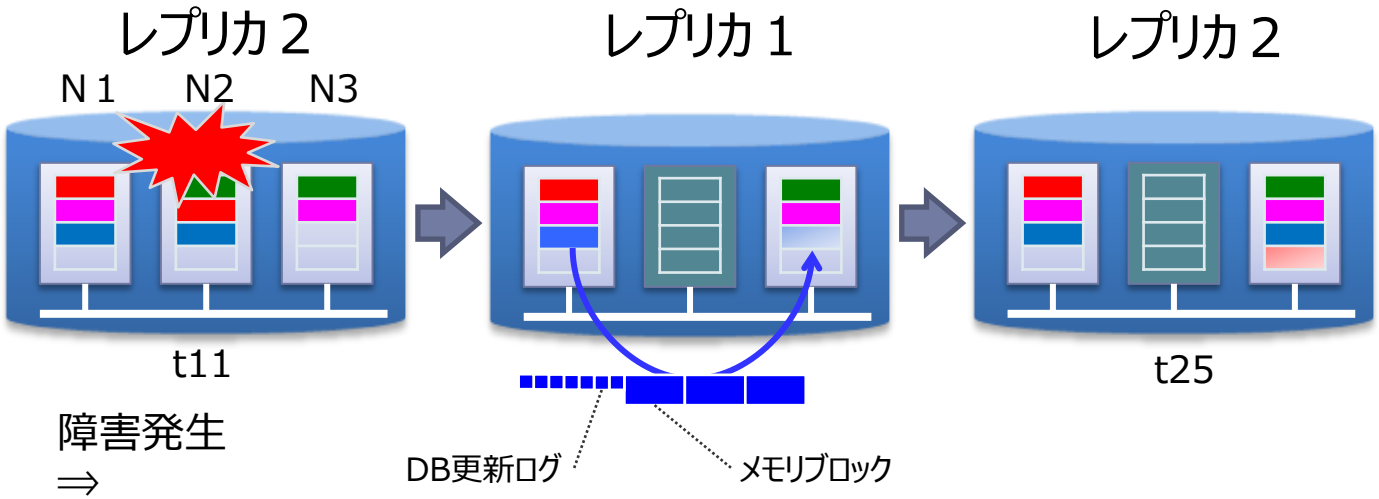
④ 高スケーラビリティ

自律データ再配置技術（ＡＤＤＡ：Autonomous Data Distribution Algorithm）

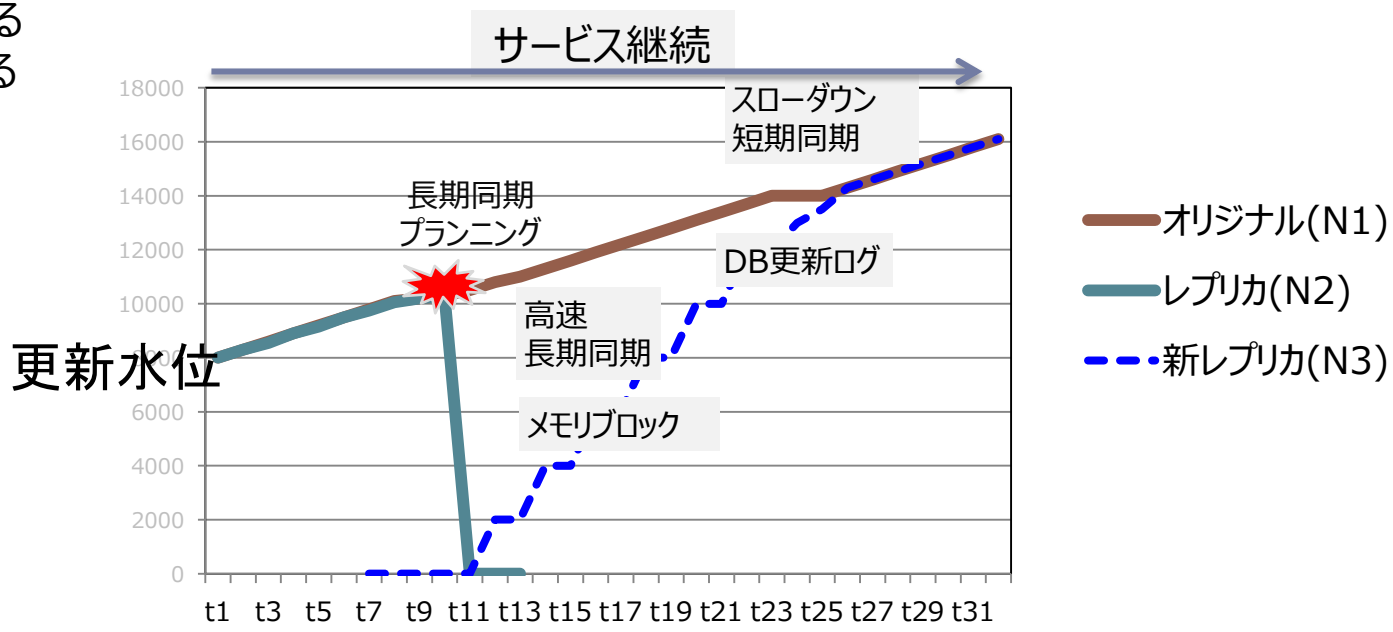
- インバランス状態を検知、長期同期プランニング
- ２種類のデータを使ってバックグラウンド高速同期、完了後切替

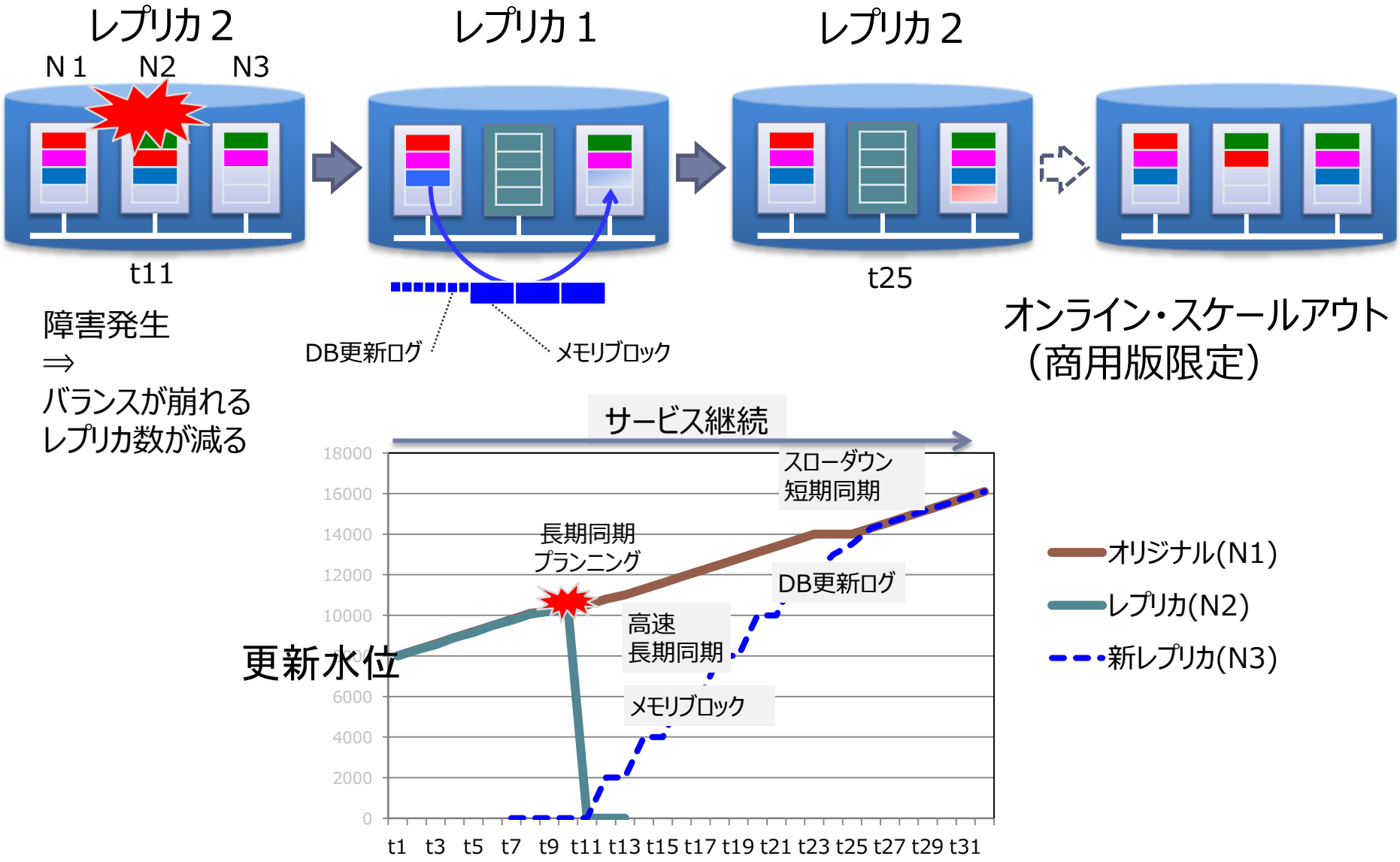


①負荷インバランス検知　②長期同期プランニング　③長期同期実行　④アクセス切替



障害発生
⇒
バランスが崩れる
レプリカ数が減る

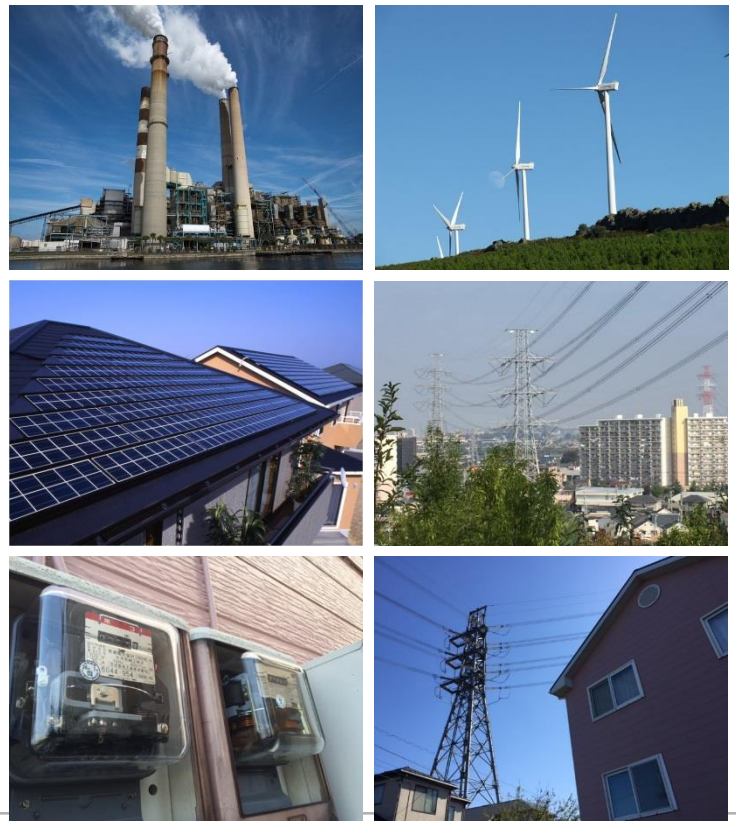


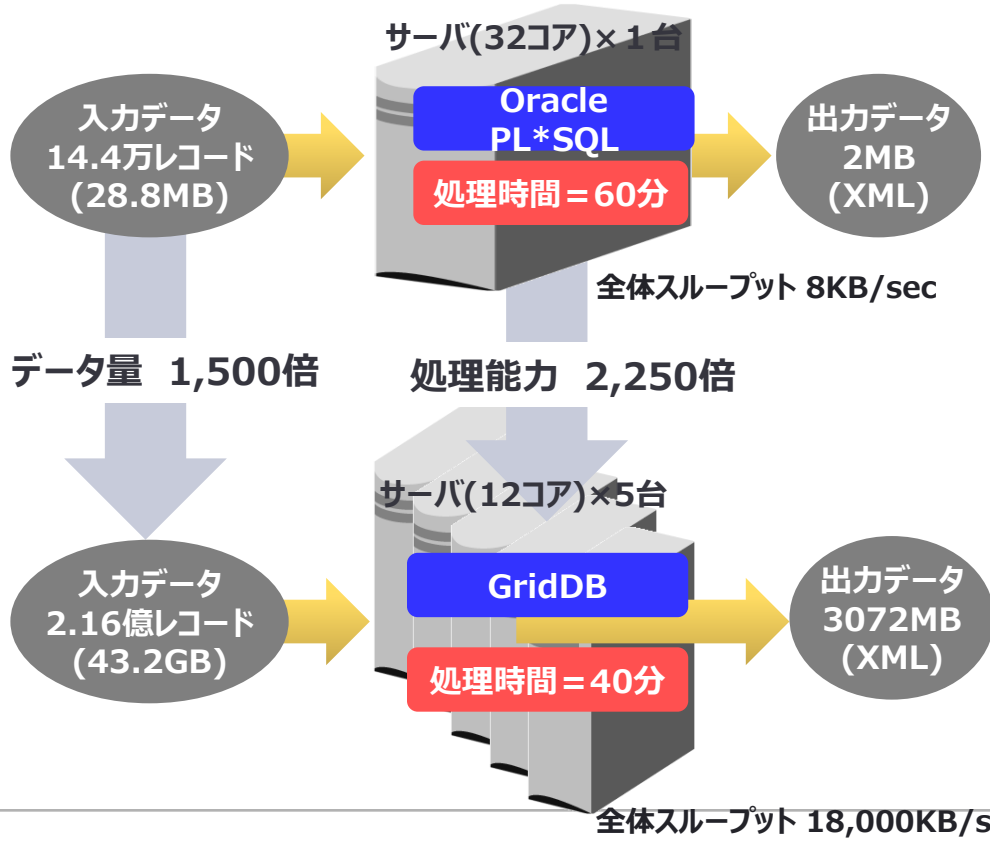

レプリカ２
N１
N2
N3
t11
障害発生
⇒
バランスが崩れる
レプリカ数が減る
レプリカ１
DB更新ログ
メモリブロック
レプリカ２
t25
オンライン・スケールアウト
（商用版限定）
サービス継続
更新水位
スローダウン
短期同期
長期同期
プランニング
DB更新ログ
高速
長期同期
メモリブロック
18000
16000
14000
12000
10000
6000
4000
2000
0
t1 t3 t5 t7 t9 t11 t13 t15 t17 t19 t21 t23 t25 t27 t29 t31
オリジナル(N1)
レプリカ(N2)
新レプリカ(N3)

適用事例

GridDB（商用版）を適用し、1,500倍のデータ量を従来比の2,250倍の処理能力で対応

電力小売り事業者に対し、電力送配電網を提供し、契約ユーザの利用量に応じた料金を請求するシステム
電力の自由化に伴い、多数の電力小売り事業者が参入し、契約数の増加（3,000契約→450万契約）による
データ量の爆発的増加へビッグデータ技術を適用し対応

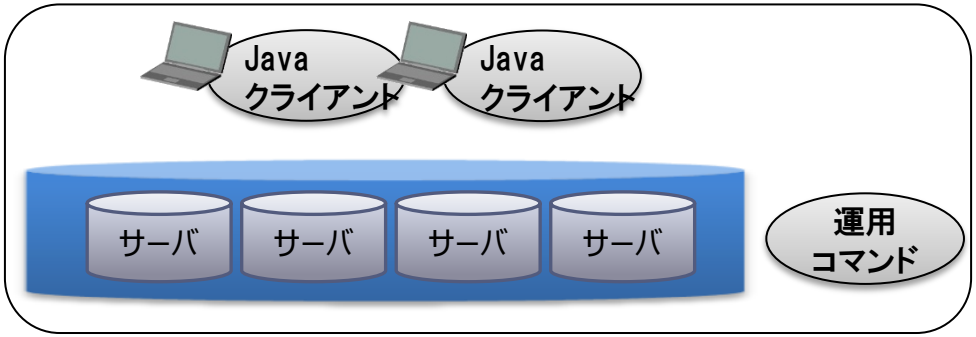


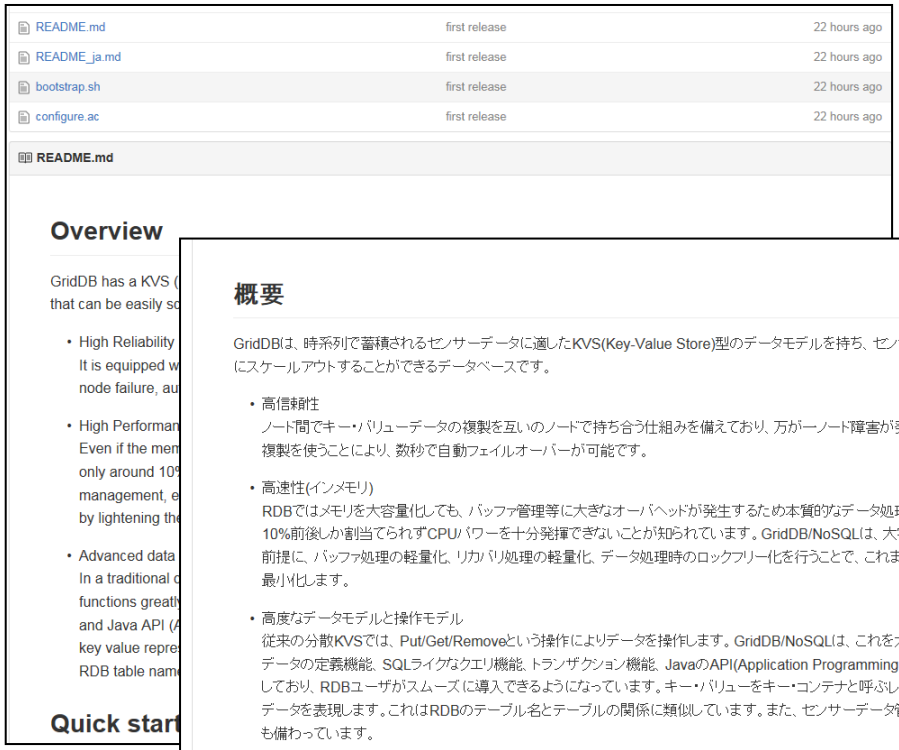


公開サイト

(2016/2時点)

- **主な公開物件**
  - ソース：
    - サーバ(C++)
    - Javaクライアント
    - 運用コマンド
  - ドキュメント（日/英）：
    - スタートアップガイド
    - 技術文書(クラスタ管理)
    - APIリファレンス
- **コミュニティ方法**
  - 質問、要望等は
    GitHubのIssueを利用





GridDBの機能概要

特徴	内容
データ構造	キーコンテナ型： 　コンテナ 　└レコード（ロウ） 　　└カラム 型：文字列、数字、真偽値、小数、時刻、バイナリ、配列
クエリ・インデックス	クエリ：SELECT、集計、サンプリングなど インデックス：文字列、数字、真偽値、小数、時刻型のカラム
API	Java Native API、SQLライクな言語(TQL) HTTP(JSON)
水平分散	ハイブリッド型、自動負荷分散あり
レプリケーション	オーナ（オリジナル）、バックアップ（レプリカ）
その他	永続化（インメモリ／ディスク併用） コンテナ単位のトランザクション 時系列データ管理、期限解放 トリガ、など

目次

起動、サンプル実行（１台構成）

- **サーバの起動**

```
$ export GS_HOME=$PWD
$ export GS_LOG=$PWD/log
$ bin/gs_passwd admin
  #input your_password
$ vi conf/gs_cluster.json
  # "clusterName":"your_clustername" #<-- input your_clustername
$ bin/gs_startnode
$ bin/gs_joincluster -c your_clustername -u admin/your_password
```

環境変数の設定

パスワードの設定

クラスタ名の設定

起動、クラスタ構成

- **サンプルプログラムの実行**

```
$ export CLASSPATH=${CLASSPATH}:$GS_HOME/bin/gridstore.jar
$ mkdir gsSample
$ cp $GS_HOME/docs/sample/program/Sample1.java gsSample/.
$ javac gsSample/Sample1.java
$ java gsSample/Sample1 239.0.0.1 31999 your_clustername
    admin your_password
--> Person: name=name02 status=false count=2 lob=[65, 66, 67, 68,
69, 70, 71, 72, 73, 74]
```

ビルド

実行

ブロードキャスト用の
IPアドレス、ポートNo

ディレクトリ構成

```
bin/    //運用コマンド、モジュールディレクトリ
        gsserver                // サーバ実行ファイル
        gridstore.jar           // Javaクライアント
        gs_startnode            // 運用コマンド
        …
conf/   // 定義ファイルディレクトリ
        gs_node.json            // ノード定義ファイル
        gs_cluster.json         // クラスタ定義ファイル
        password                // ユーザ定義ファイル
data/   // データベースファイルディレクトリ
        gs_cp_xxx               // チェックポイントファイル
        gs_log_xxx.log          // トランザクションログ(Redoログ)
log/    // イベントログ出力ディレクトリ
        gridstore_xxx.log       // イベントログファイル
                                    （トレース出力、エラー出力）
```

設定手順

（A）初期設定

1. 環境変数(GS_HOME, GS_LOG)
2. ユーザアカウントの作成(gs_adduser, gs_passwd, gs_deluser)
3. クラスタ名の設定(conf/gs_cluster.jsonのclusterName)

（B）パラメータ設定(conf/gs_node.json)

1. メモリ上限値(dataStore/storeMemoryLimit)
2. 並列度(dataStore/concurrency)

（C）クラスタ構成時の設定

1. ノード数(gs_joinclusterの-nオプション)
2. ブロードキャスト用のIPアドレス、ポートNo
   (conf/gs_cluster.jsonのtransaction/notificationAddress,
   transaction/notificationPort)

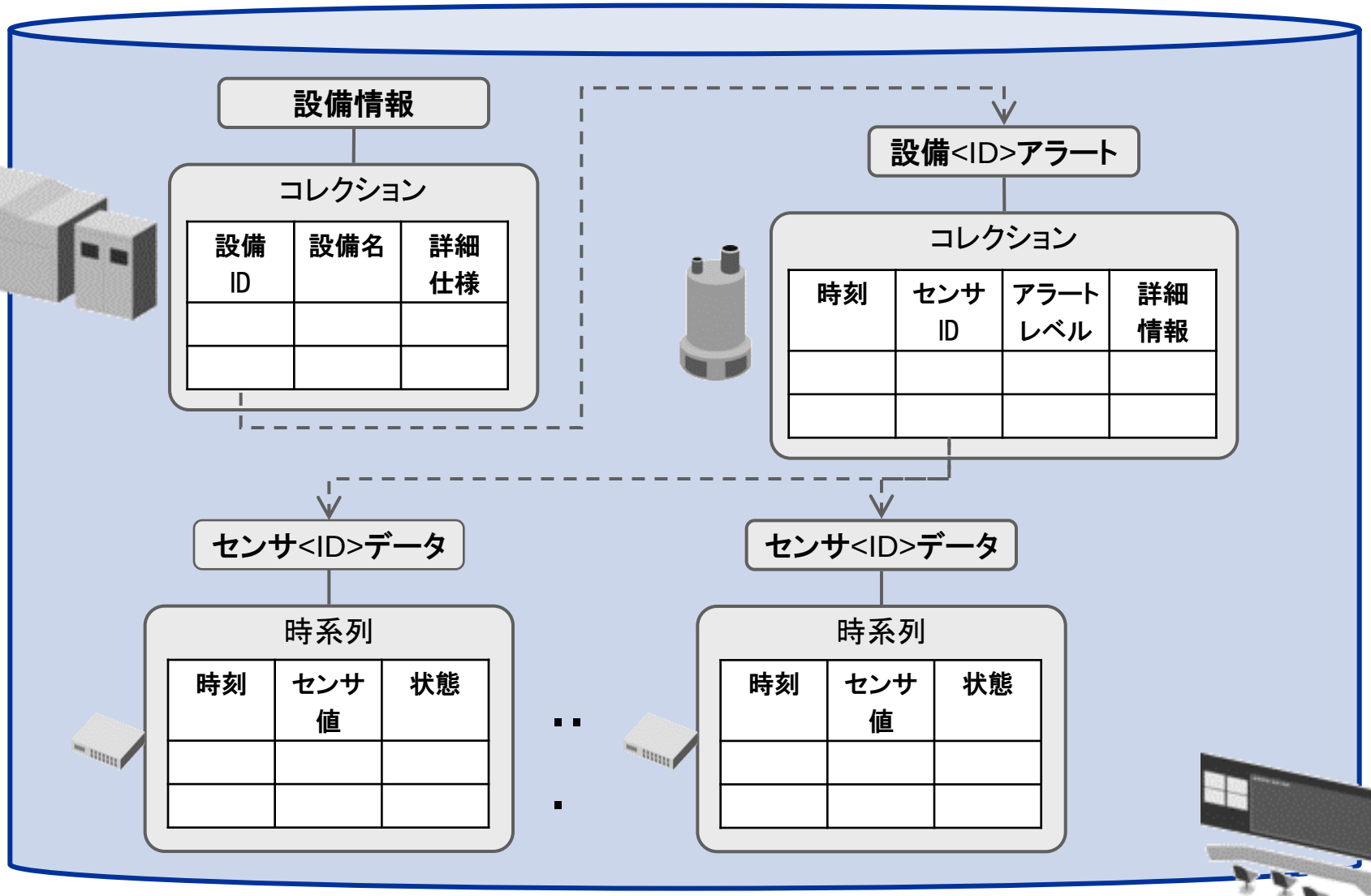
運用コマンドの実行手順

1. **起動(各ノードについて)**
   - 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 接続サーバ: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 接続サーバ: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 接続サーバ: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 接続サーバ:ポート] -u ユーザ名/パスワード

プログラミングの例


設備情報
コレクション
設備 ID
設備名
詳細 仕様
設備<ID>アラート
コレクション
時刻
センサ ID
アラート レベル
詳細 情報
センサ<ID>データ
センサ<ID>データ
時系列
時刻
センサ 値
状態
時系列
時刻
センサ 値
状態

レコードデータの登録

```
// アラート情報
class Alert {
    @RowKey long id;
    Date    timestamp;
    String  sensorId;
    int     level;
    String  detail;
}
```

```
final GridStore store = GridStoreFactory.getInstance().getGridStore(prop);
// コレクション登録
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];

    // 値登録
    alertCol.put(alert);
```

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);
…
// 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 > " + from );

RowSet<Alert> alertRs = alertQuery.fetch();
while( alertRs.hasNext() ) {
    // 重大アラート発生したセンサ
    Alert alert = alertRs.next();
    // 直前の時系列データを検索
    TimeSeries<Point> ts = store.getTimeSeries(alert.sensorId, Point.class);

    Date endDate   = alert.timestamp;
    Date startDate = TimeStampUtil.add(alert.timestamp, -10, TimeUnit.MINUTE);
    RowSet<Point> rowSet =  ts.query(startDate, endDate).fetch();

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

目次

まとめ

- **GridDBは高速処理と高信頼性を両立し、ペタバイト級の多種大量データを蓄積する、ビッグデータ/ＩｏＴ時代のデータベースです。**
  - High Performance
  - High Scalability
  - High Reliability

GridDBはオープンソースになったので、是非ともご覧ください、使ってみてください。


TOSHIBA
Leading Innovation