

オープンソースカンファレンス 2013 Nagoya

Cassandra , HBase で学ぶ NoSQL 入門

2013年6月21日

盛 宣陽

y-mori@sraoss.co.jp

SRA OSS, Inc. 日本支社
マーケティング部

NoSQLとは

- NoSQL(Not only SQL)とは
RDBMS以外のデータベースを示す総称
- なにかしらの点でRDBMSより性能がよいデータベース

性能とは？

レスポンス、スループット、対障害性、スケーラビリティ

- RDBMSの機能を犠牲にして
要件に合わせた性能を満たすデータベース

RDBMSのデータモデルと特性

- RDBMSではリレーショナルモデルを採用
 - 正規化によりデータを一元管理
 - リレーシヨンの結合により論理データを扱う
- RDBMSではACID特性を満たす
 - A(原始性) トランザクションは成功 or 失敗で終わる
 - C(一貫性) データに矛盾が生じない
 - I (独立性) 他のトランザクションから影響されない
 - D(耐久性) 障害が発生しても、更新結果は保持される。

一般にNoSQLでは、リレーショナルモデルとACIDを排除することで、性能を満たす

NoSQLのデータモデル

NoSQLの種類とデータストアの種類

- **key-value型**

ユニークなキーに対してバリュー(値)を持つ。

NoSQLデータベースの基本的なデータ構造

例) memcached, Redis

- **カラム指向型**

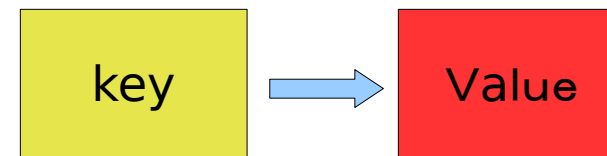
キーに対してカラム(名前と値)を複数持つことができる

HBase, Cassandra

- **ドキュメント指向**

XMLやJSONと言ったスキーマレスでデータ構造を持つ

MongoDB



NoSQLとBASE

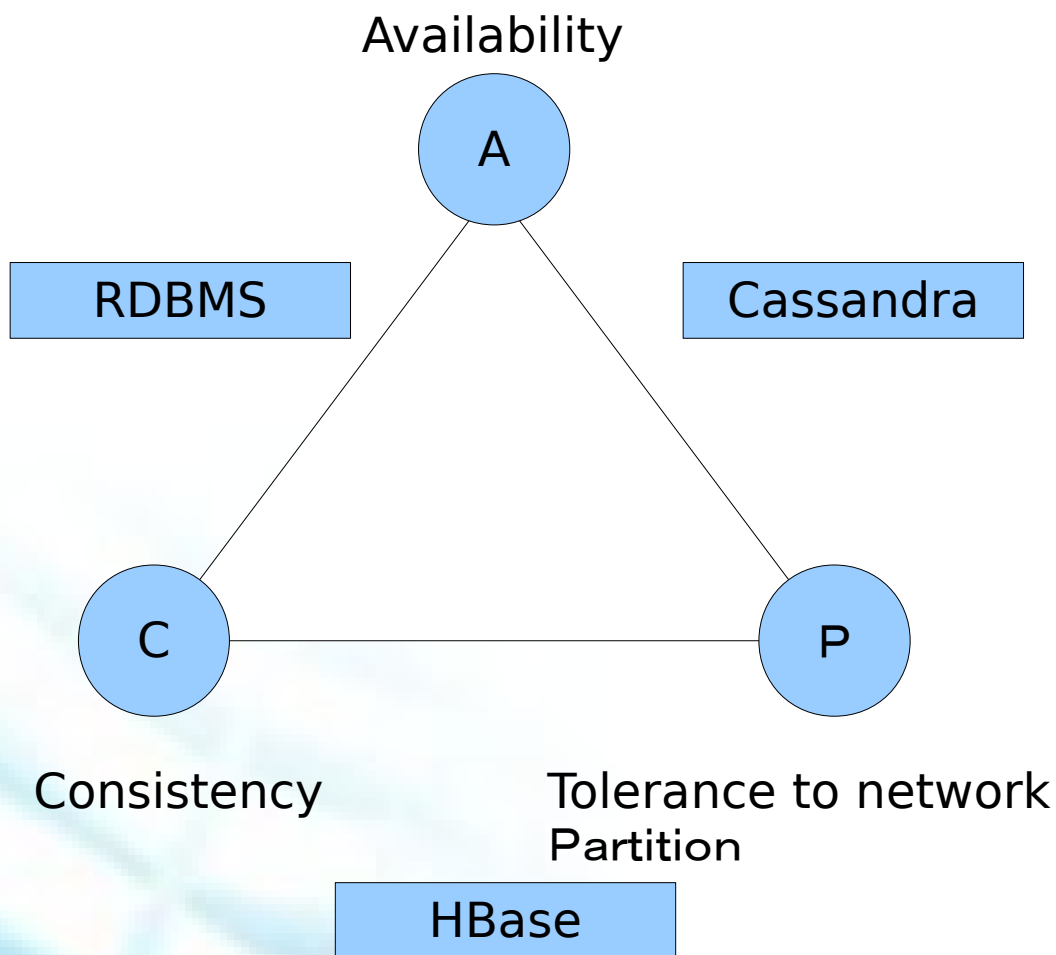
- トランザクションの性質 [BASE]
 - BA (Basically Available)
 - 可用性が基本 (サービス維持が基本)
 - 部分障害への対応
 - S (Soft State)
 - 厳密ではない状態遷移
 - E (Eventual Consistency)
 - 結果整合性

NoSQLの特性

- key-valueによってKey単位で分散処理が可能
スケールアウトに対応
 - 分散ハッシュテーブルの利用
- ACIDを排除
 - 性能を妨げる強いロックを削減
 - データ間の依存関係をなるべく削除
(MVCC非採用)
 - 高速なレスポンス

クラウド環境やビッグデータ時代にマッチ

分散データベースとCAP定理(ブリュワーの定理)



- **C**onsistency
すべてのクライアントから同じ結果が見える(整合性)
- **A**vailability
すべてのクライアントがシステムを常に利用できる(可用性)
- Tolerance to network **P**artition
ネットワークが分断されても、システムが稼働する

CAP定理

CAPのうち同時に二つしか満たすことができない

HBase
<http://hbase.apache.org/>

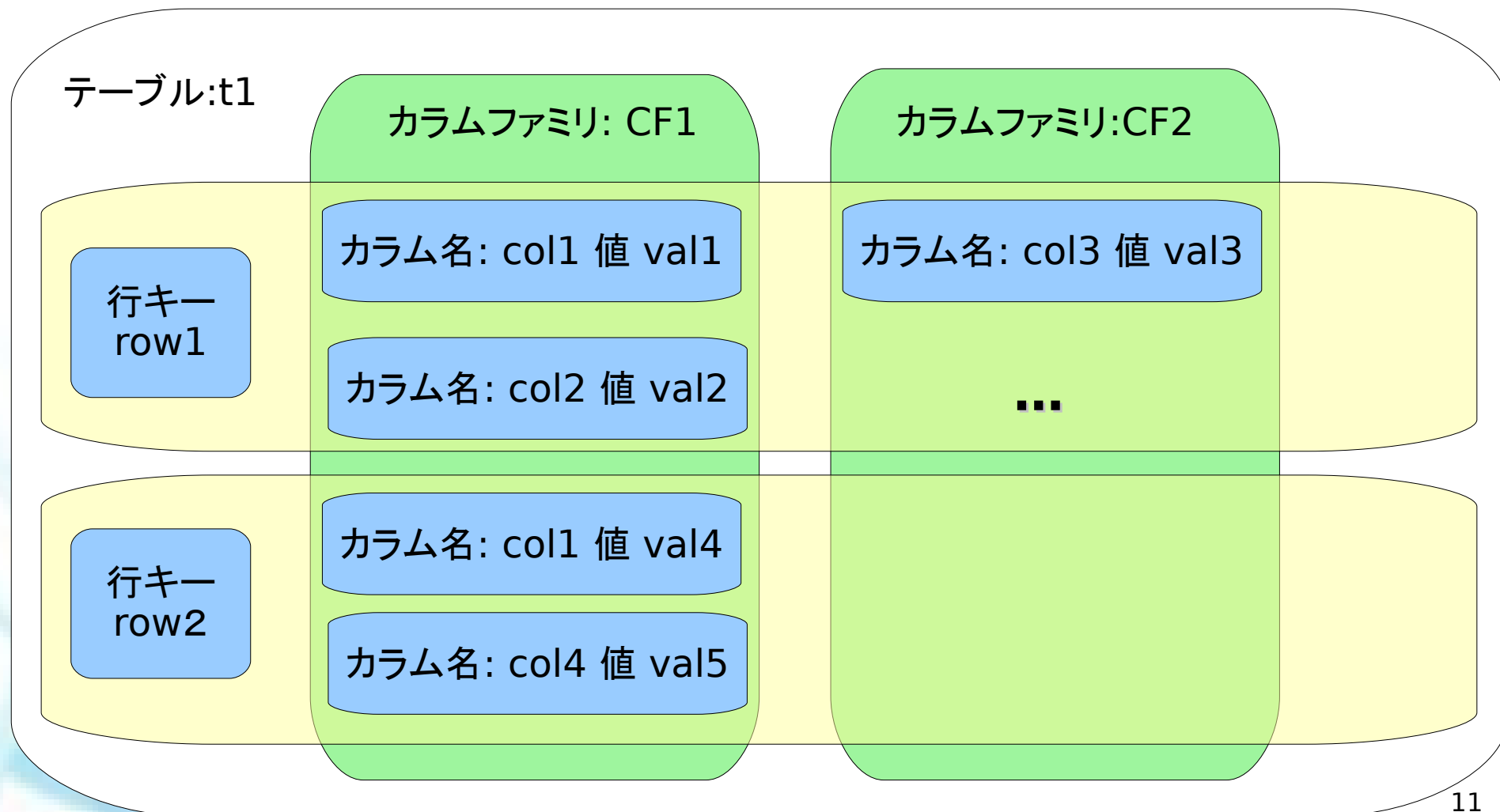
HBaseとは

- 分散データストアシステム (カラム指向)
- ビッグデータ向け (テラバイト規模)
- Javaベース
- 行レベルの一貫性
- 線形スケーラビリティ
- 自動論理分割(シャーディング)
- Hadoop分散ファイルシステム(HDFS)上に構築
レプリケーションはHDFSの機能に依存
- Apache License Version 2.0

HBase データモデル

データモデル

- カラム指向



テーブルの作成

- 以下の図で表されるスキーマを定義する

テーブル:test		
行キー	カラムファミリ	カラム
row1	colfam1:	カラム:a
	colfam2:	カラム:b, c

- createコマンドを使用してテーブルを作成

```
hbase(main):002:0> create 'test', 'colfam1', 'colfam2'  
0 row(s) in 1.2070 seconds
```

データの追加と取得

- putで行と新しい列を追加

```
hbase(main):003:0> put 'test', 'row1', 'colfam1:a', 'data1'  
0 row(s) in 0.1200 seconds  
hbase(main):004:0> put 'test', 'row2', 'colfam2:b', 'data2'  
0 row(s) in 0.0110 seconds
```

高度な機能

- バージョン (タイムスタンプ)
 - セルごとにバージョン(タイムスタンプ)をもつ
- TTL (生存期間)
 - 列ファミリごとにTTL(生存期間)を決められる
- フィルタ機能
 - 文字列や数値の比較、正規表現などにより結果をフィルタリングできる

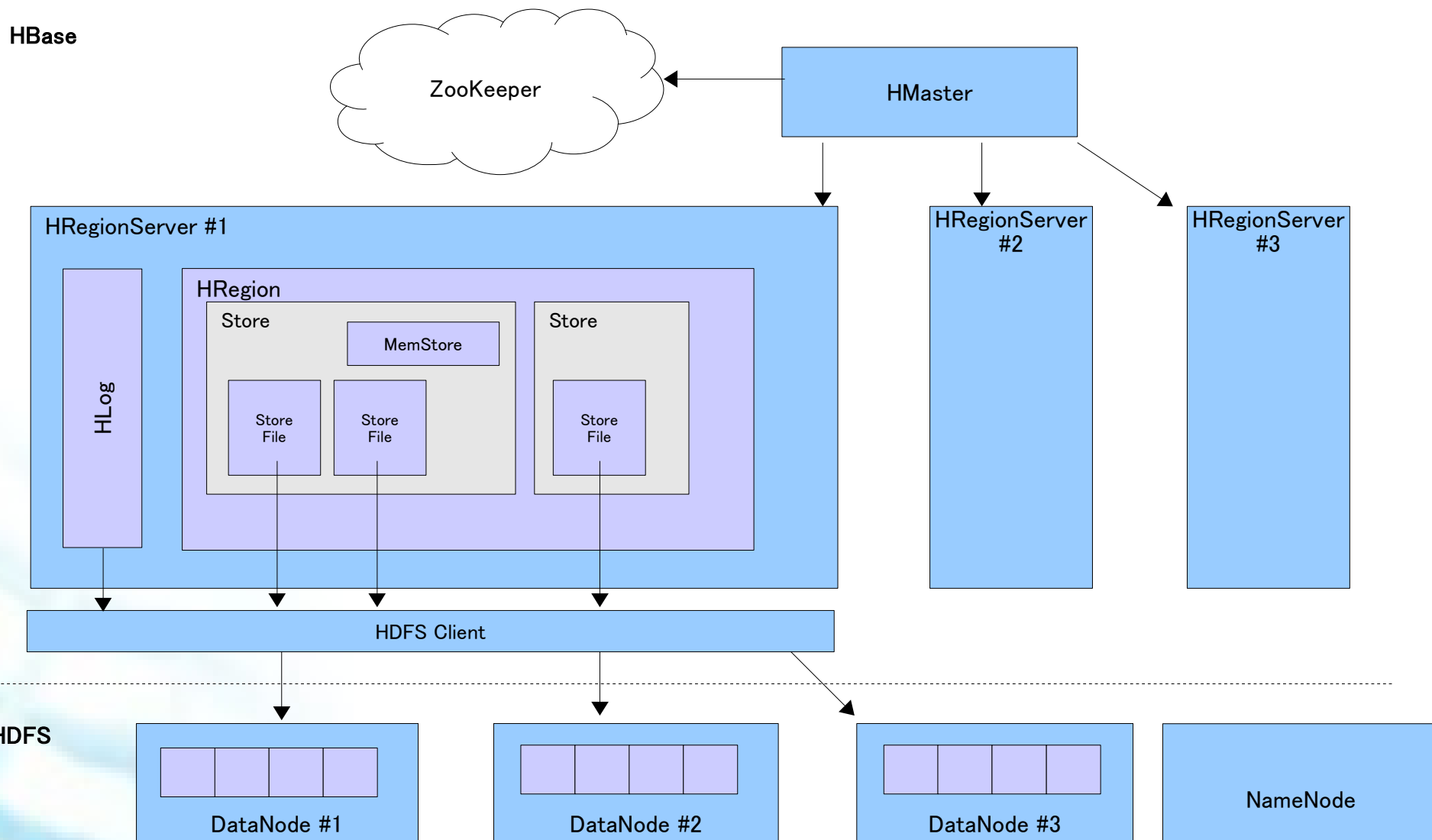
```
hbase(main):028:0> scan 'test', {FILTER => "(PrefixFilter ('row2')
AND ValueFilter(=, 'binary:data2'))"}
ROW          COLUMN+CELL
row2        column=colfam2:b, timestamp=1342511049557, value=data2
```

HBase概要

HBaseを構成するサーバ

- HBase Master
- リージョンサーバ
- ZooKeeper
- Hadoop (HDFS)
 - NameNode
 - DataNode

HBaseの構成図



リージョン (概念図)

- 1つのテーブルは複数のリージョンに自動分割される

- 分割の単位は連続した範囲の行キー

例) 行キーの先頭が A~M と

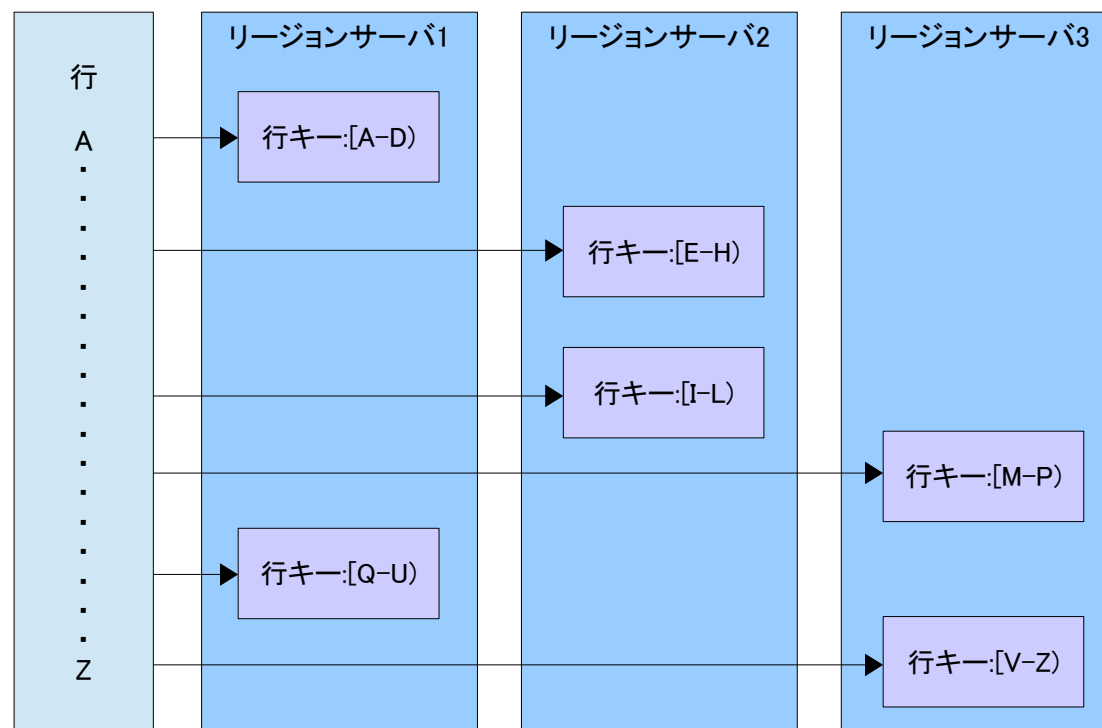
N~Z で分ける

- リージョンは複数のリージョンサーバに分散される

- 1つのリージョンサーバは複数のリージョンを扱う

- リージョンサーバを増やすことで、処理はスケールする

- 行キーは特定のリージョンサーバで管理されるため、行キーレベルの一貫性を保つ

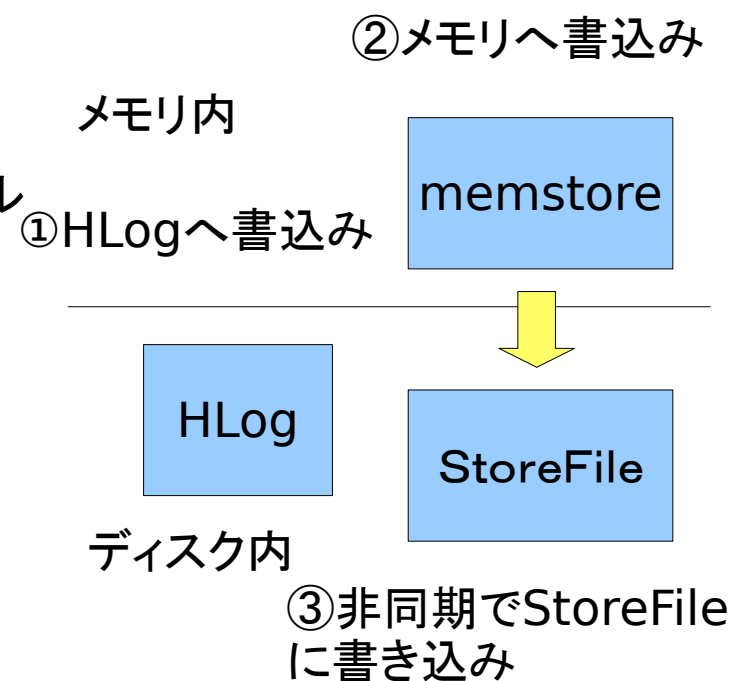


リージョンサーバ 書込み処理

- 書込み要求はHLog(トランザクションログ)に記述してからメモリ上のMemstoreに保存される
- Memstoreが溢れるとカラムファミリー単位でStoreFileに出力して永続化を行う

重要 書込みルール

- 既存ファイルは上書きしない
同じキー、カラムファミリーのデータは複数ファイルで管理(書込み直後にはマージされない)
- ファイルは行キーでソートされている



リージョンサーバ 読み込み処理

- Memstoreに要求データがあればそのままクライアントにデータを返却
- Memstoreに無ければ、StoreFileから対象データを探す
 - ここで、特定のキーのデータは複数のStoreFileから探すことになる
(注)一つのカラムファミリのデータは各リージョンサーバ内の複数のStoreFileに存在する

HBase Masterの役割

- リージョンを各リージョンサーバに割り当てる
- 負荷分散 (リージョンのサーバ間移動)
- クラスタの状態の管理
(リージョンサーバの自動フェイルオーバーとリージョンの再割り当て)
- スキーマの作成、変更
- クライアントとは直接のやりとりは行わない

ZooKeeperの役割

- クラスタの各ノードの情報の管理
- 最初にクライアントからの接続を受け付ける
- 「-ROOT-」システムテーブル(メタデータ)を格納するリージョンの場所をクライアントに通知する

リージョンの場所を受け取ったクライアントは、直接リージョンに接続を行い、通信を行う

Hadoopの役割

- HDFS (Hadoop 分散ファイルシステム)を提供
- HBaseで扱うデータは実際にはHDFS上に格納される
- HBase自体にはレプリケーションの機能はなくて、HDFSによってデータのレプリケーションが行われる
- HDFSは、バックグラウンドでノード間のデータ比較を行う
破損しているデータがある場合には自動復旧される

障害ケース①

- HBase Master

- 短時間の停止はシステム全体の停止に直結しない
- 長時間の停止はクラスタの破綻につながる
- 対策

HBase Masterのホットスタンバイ構成

=> Zookeeperと連動し自動フェイルオーバーが行われる

- リージョンサーバ

- 担当しているデータは一時的に利用不可となる
- マスタノードによってリージョンのフェイルオーバーが自動的に行われる
- クライアントが障害を検知した場合には、マスタノードに通知を行い、マスタノードがリージョンをフェイルオーバーさせる

リージョンの移動後、クライアントはデータアクセスを行う

障害ケース②

- Zookeeper
 - 複数台で運用するためクライアント側で接続先を切り替える
- HDFS
 - NameNodeが単一障害点となる
 - 一般的にHAクラスタで冗長化

Hadoop 2系でNameNodeがActive-Standby方式のHA機能が追加される(現在alpha版)

- ネットワーク分断

ZookeeperがHBaseクラスタのノード情報を比較し、多数決によって稼働を続けるノードを決定する

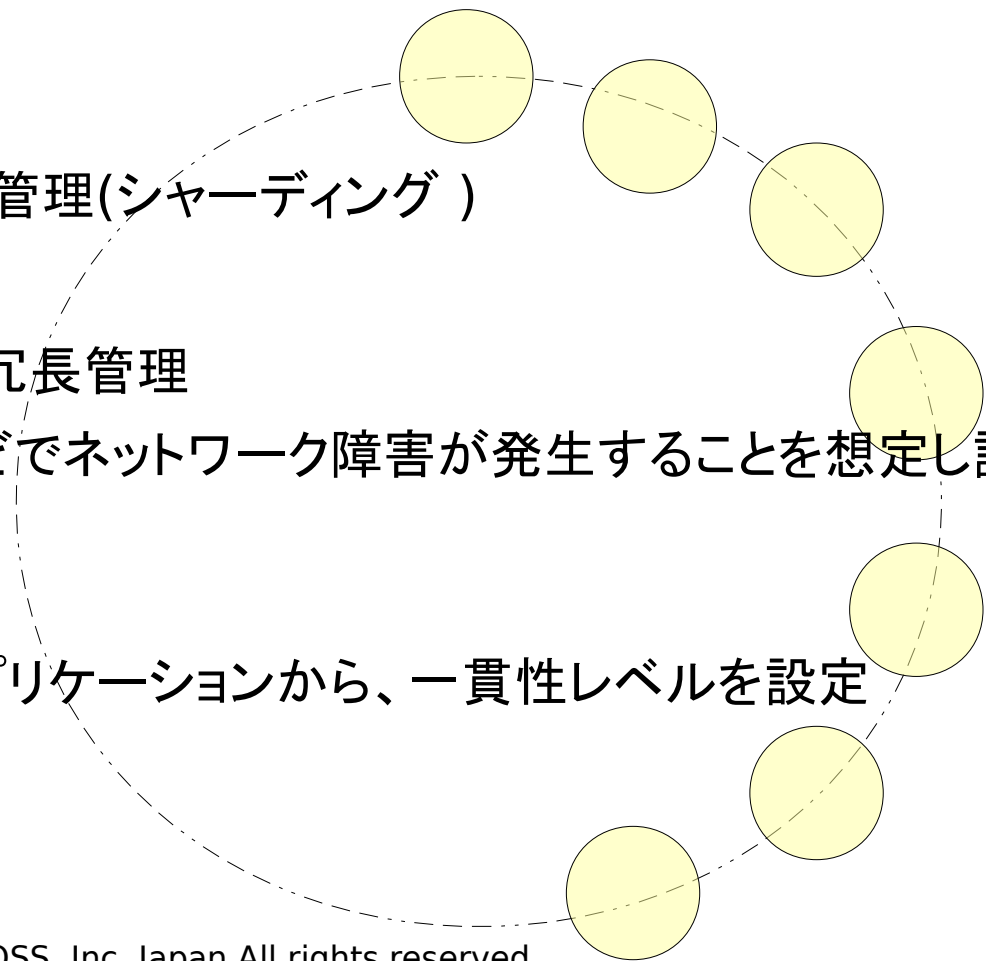
そのためZookeeperは奇数台が推奨される



<http://cassandra.apache.org/>

Cassandraとは

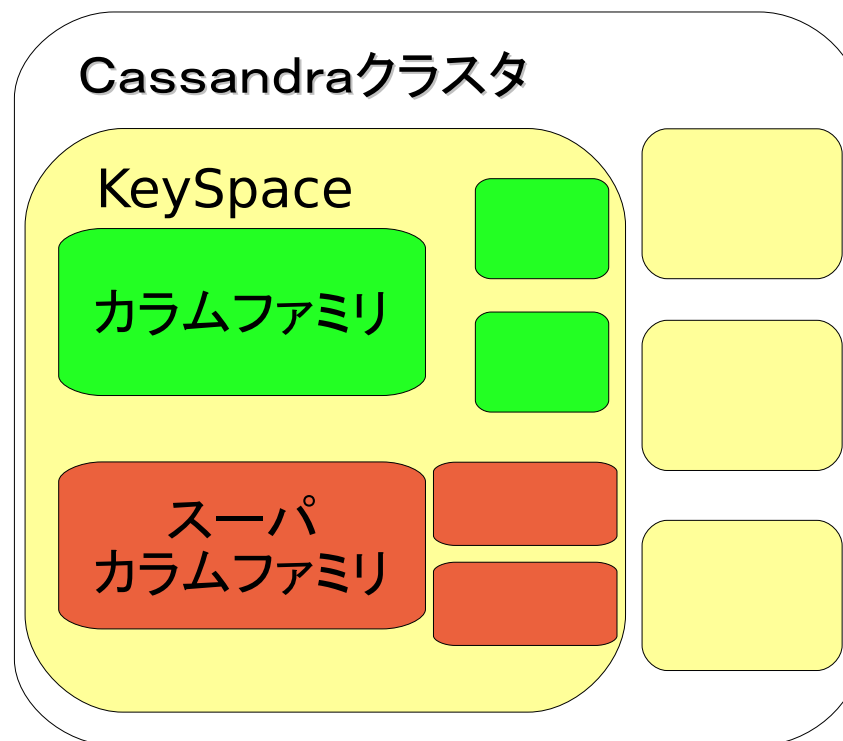
- マルチマスタ
 - 単一障害点がなく、すべてのノードが同じ役割をする
 - すべてのノードで読み書き可能
- スケーラビリティ
 - データを複数のノード間で分散管理(シャーディング)
- 高可用性と対障害性
 - 複数のノード間で同じデータを冗長管理
 - ラック間、データセンター間などでネットワーク障害が発生することを想定し設計されている
- 調整可能な一貫性レベル
 - Cassandraに接続を行うアプリケーションから、一貫性レベルを設定
- Apache License Version 2.0
- Javaベース



Cassandraデータモデル

Cassandra スキーマ

- keySpace
もっとも外側のデータ格納領域
RDBMSのデータベースに相当
複数作成可
- カラムファミリー or スーパーカラムファミリー
論理データの集合
RDBMSのテーブルに相当
複数作成可



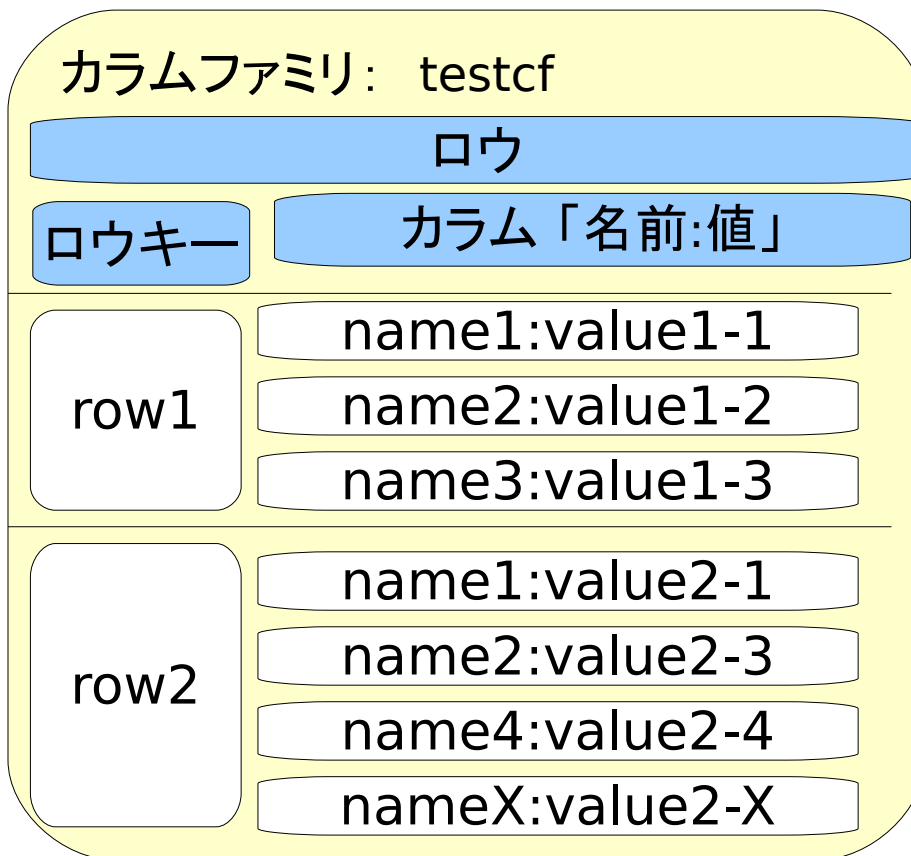
カラムファミリー

- カラムファミリーは複数の「ロウ」で構成
- 各ロウには、ユニークな「ロウキー」を指定する
- 各ロウには、ロウキーに従属した複数の「カラム」により構成される
- カラムには「名前:値」が格納される
- [カラムファミリー名][ロウキー][カラム名]を指定して値にアクセス

(注)

各ロウに入っているカラムの名前は異なってもよくデータ格納時に

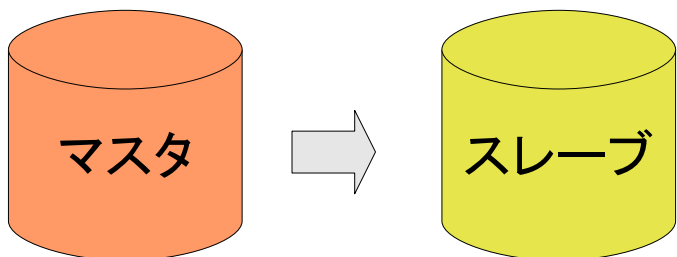
名前を決定できる Copyright © 2013 SRA OSS, Inc. Japan All rights reserved.



Cassandra概要

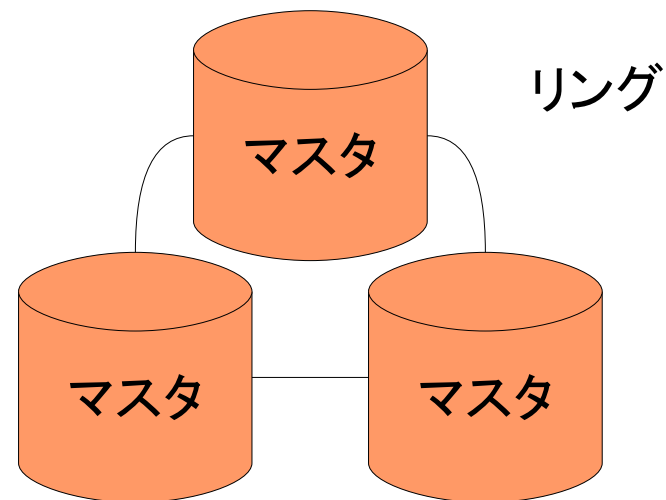
マルチマスタ

RDBMSで一般的な
マスタスレーブ構成



VS

Cassandraのマルチマスタ構成



一般的なRDBMSにおける
クラスタ構成

書き込みはマスタで
読み込みはマスタ・スレーブ

書き込み、読み込みを
アプリケーションで切り替える

Cassandraでは

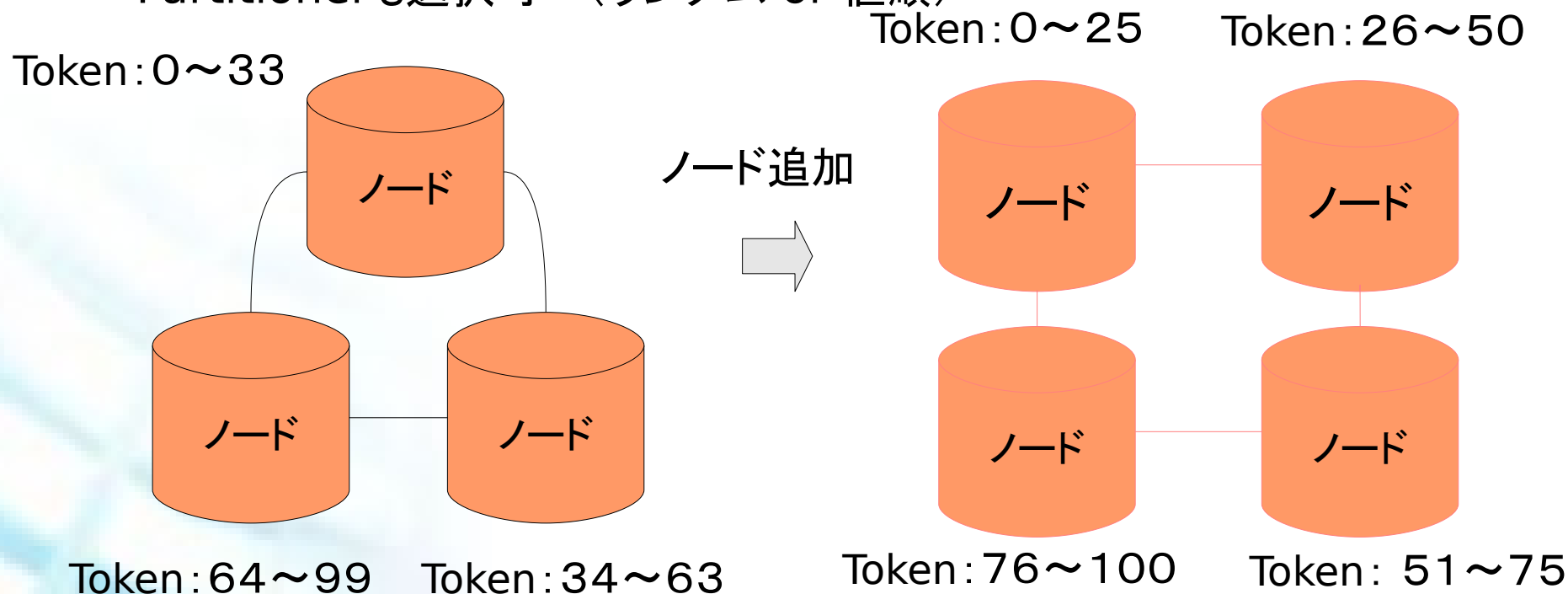
すべてのノードが対等な関係で
読み書き可能

すべてのノードが振り分け先
を決める(proxyとして機能)

スケーラビリティ

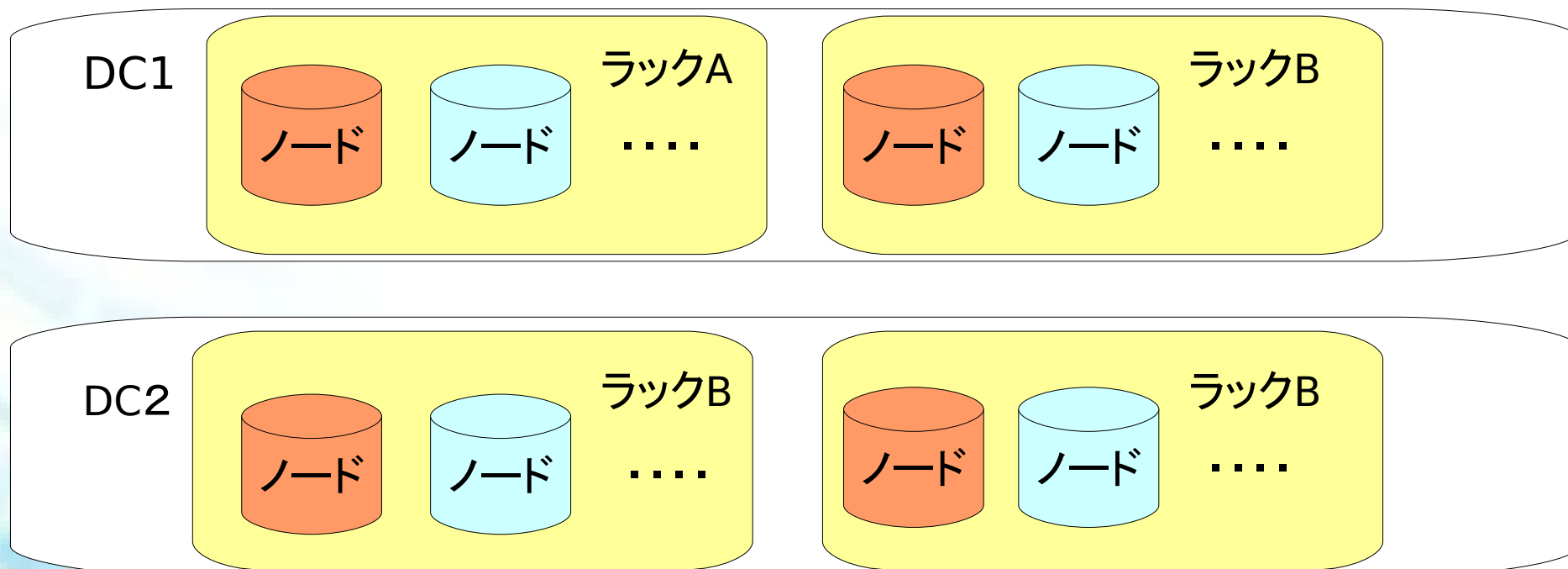
- ダウンタイムなしにノードの追加が可能
- 各ノードに担当データ範囲(Token)を設定しデータを格納
キーをハッシュ化して、Tokenと比較し担当ノードが決定される

Partitionerも選択可 (ランダム or 値順)



レプリケーション

- データの冗長化
 - キースペース単位でレプリカ数を決定
 - 隣のTokenを持っているノードがレプリケーション先となる
 - DCやラックを考慮したレプリカ戦略も可能



一貫性レベル①

- WRITE/READを行うときにアプリケーションで一貫性を調整

指定するパラメータ

ONE,TWO,THREE,QUORUM,ALL,LOCAL_QUORUM,EACH_QUORUM

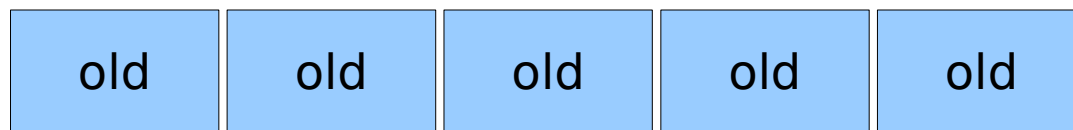
ANY (WRITEのみ)

- WRITEのときに
 - レプリケーション数(N)に従って書込みを行うときに書込み完了を待つノード数を指定する。書込み完了を待つノード数をWとする
 - 例) レプリケーション数 5 QUORUM(過半数)の場合には、3台の書込み完了を待ち、残りの2台は非同期でレプリケーションが行われる
- READのときに
 - レプリケーション数(N)以内で読み込みを行うノード数を指定する。
読み込みを行うノード数をRとする。
- $R + W > N$ を満たせば最新のデータが読み込めて、一貫性が保てる

一貫性レベル②

$N=5, W=3, R=3$ の場合には, $R+W > N$ ($R > N-W$) を満たすので最新データが読み込める

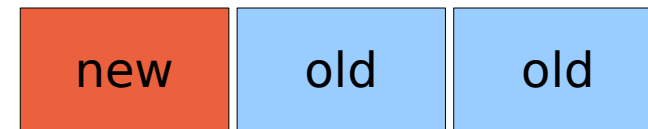
N:レプリケーションするノード数



W:書込み完了を待つノード数



R:読み込みを行うノード数

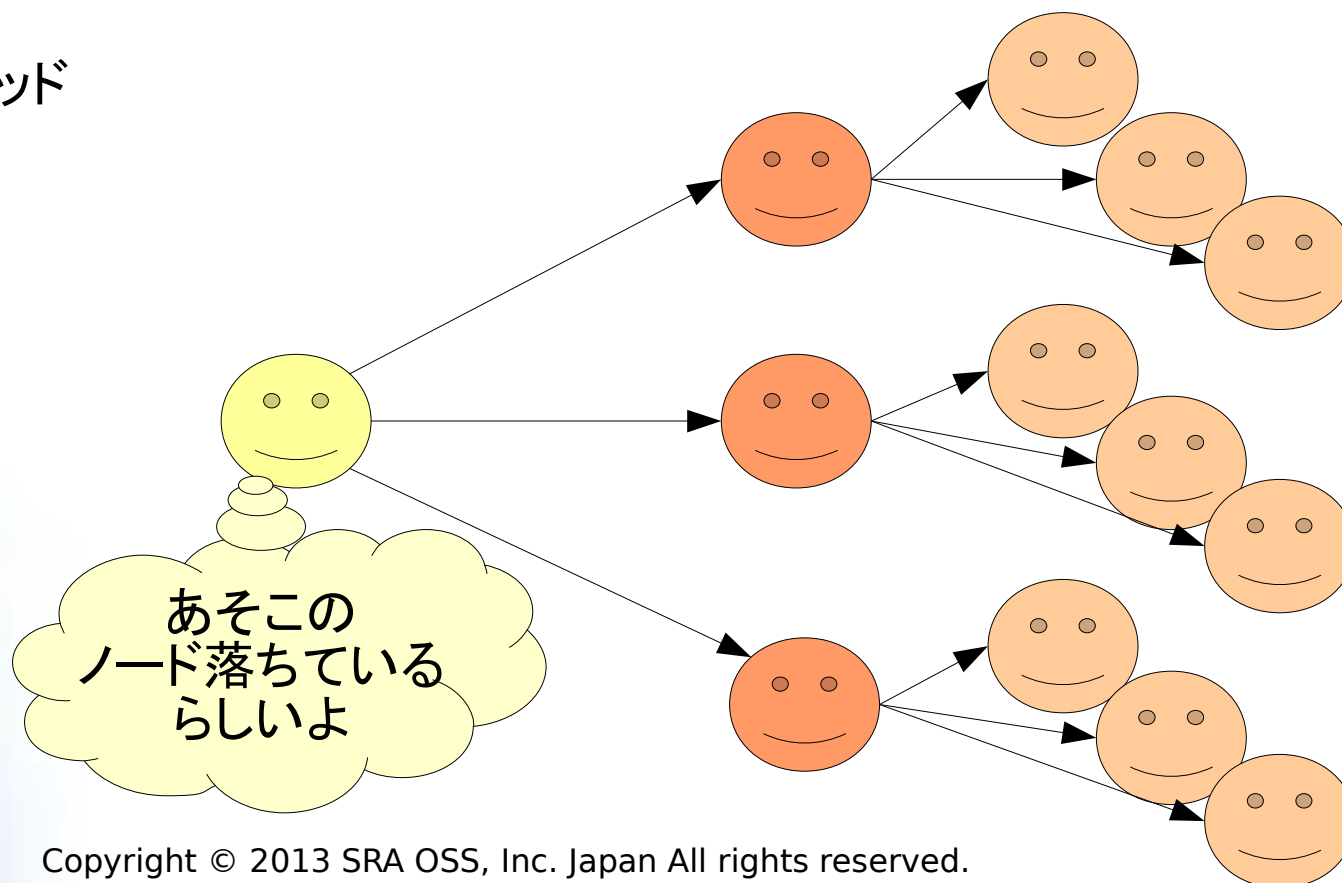


一貫性レベル③

- 注意事項
 - RDBMSのような”強い”一貫性は保てない
 - RDBMSでいうところのMVCCレベルでの一貫性は保てない
 - Cassandraでは複数のノード間で行データを取得し一貫性レベルに従って比較して返す
(行レベルの操作ですらアトミックではない)
トランザクションや、テーブル間にまたがる一貫性ももちろん保証されない
 - 各ノードの時刻に依存してデータを管理する。
NTP以上の分解能は見込めない

高可用性と対障害性①

- ノード情報の伝播は Gossip プロトコル
 - 障害を検知したノードを起点に情報（噂）が広まるように伝播する
 - 低オーバーヘッド

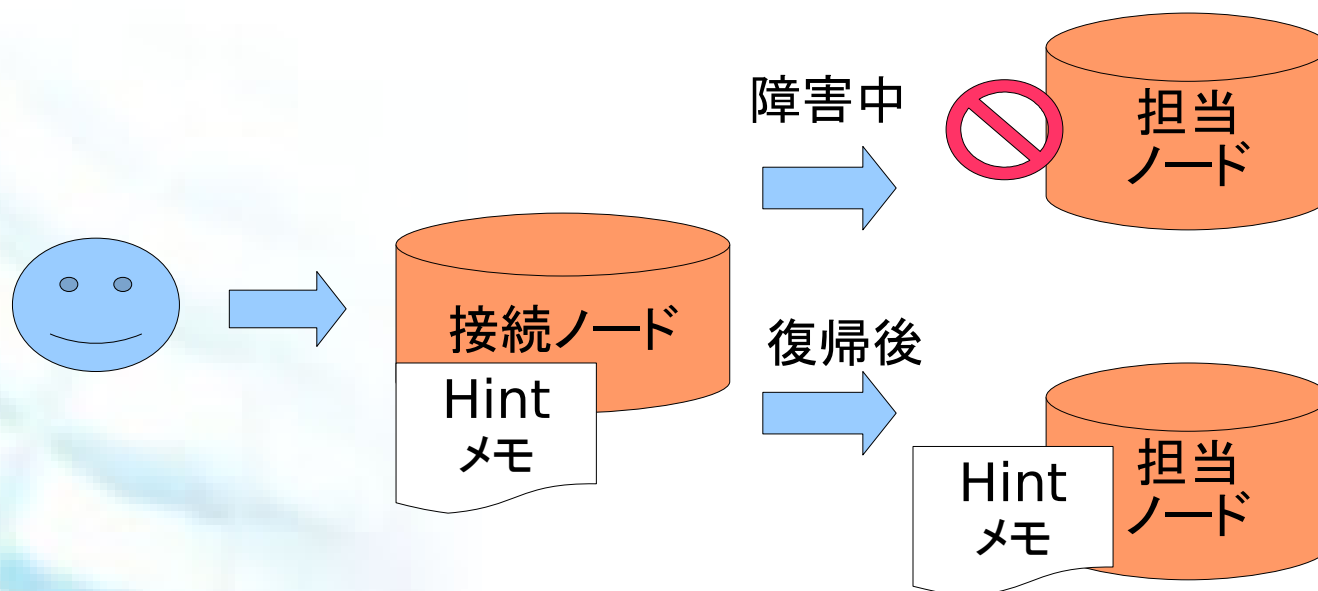


高可用性と対障害性②

- ネットワーク分断やノード停止を考慮
 - 障害回復時に、3つの処理によってデータ同期が行われる
 1. ヒントハンドオフ
 2. リードリペア
 3. アンチエントロピー

ヒントハンドオフ

- 担当ノードに障害が発生していたときに、処理を受け取ったノードが追加データを一定期間預かる仕組み
- 担当ノードが復帰時に自動的に同期が行われる
- ロウ単位の復旧

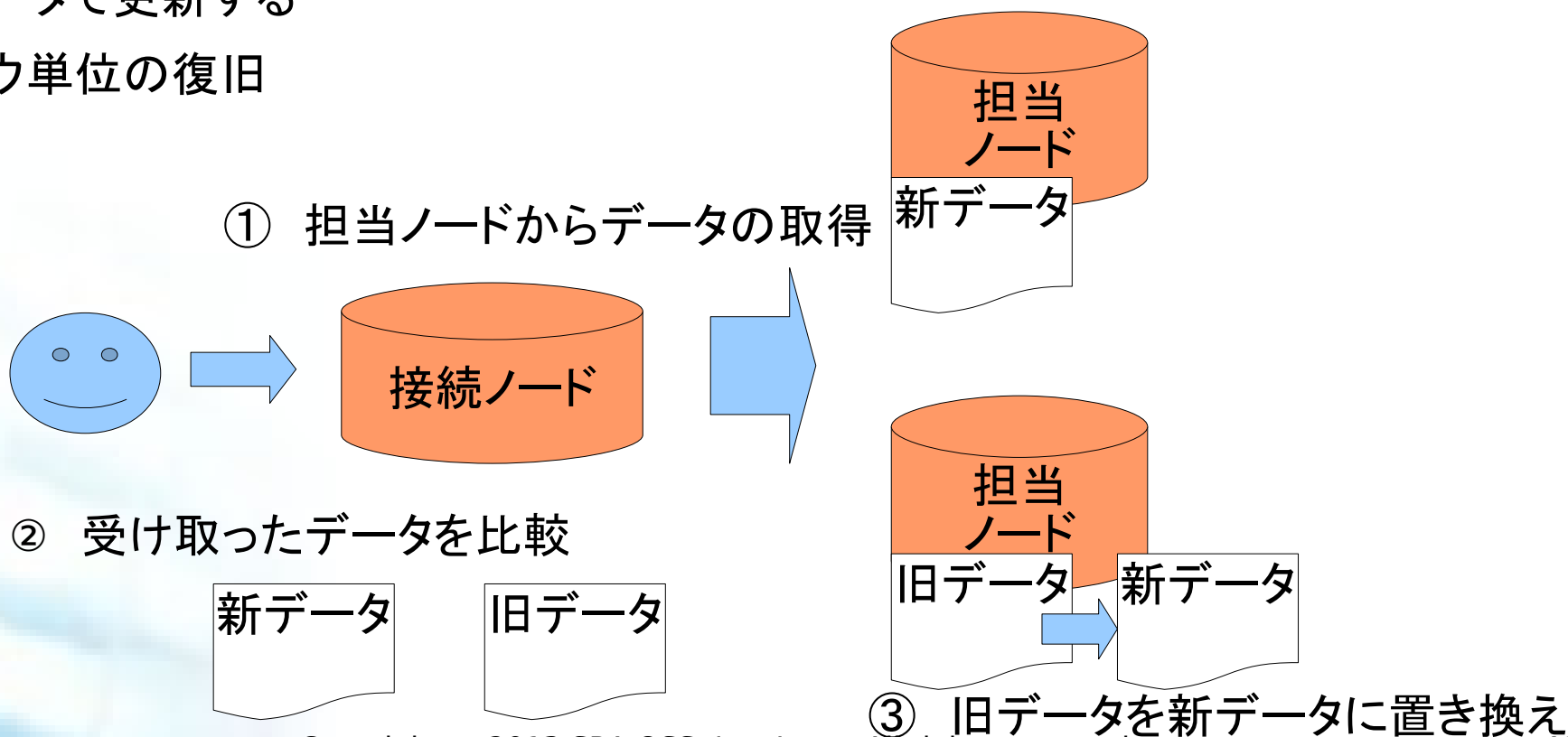


リードリペア

- READのときにデータ復旧

複数の担当ノードから返却されたデータを比較して、異なっていた場合は最新のデータで更新する

- ロウ単位の復旧

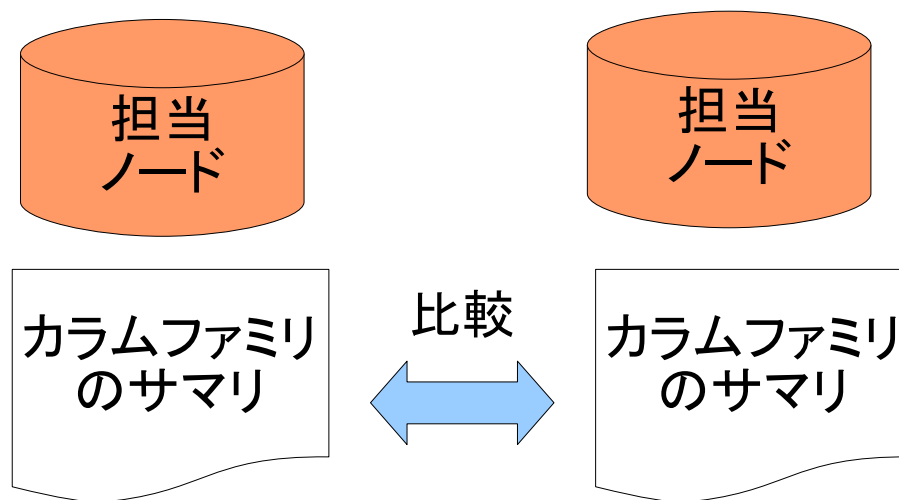


アンチエントロピー

- 各ノードがもっているデータの比較を行って結果が異なる場合にはデータ転送が行われる

ハッシュ値(マークルツリー)を利用してデータ比較を行い転送コストを削減

- カラムファミリー単位の復旧



Cassandraの内部構造

- プロセス

SEDAと呼ばれるステージ(処理)毎にイベントドリブン型のアーキテクチャ

大規模なアクセスをさばけるイベントドリブン型 + リソースをステージ毎に設定可能

- データ構造

commitlog ..RDBMSでいうところのトランザクションログ(redo log)

電源断などの復旧時はcommitlogからデータを復元

memtable ..RDBMSでいうところの共有メモリ

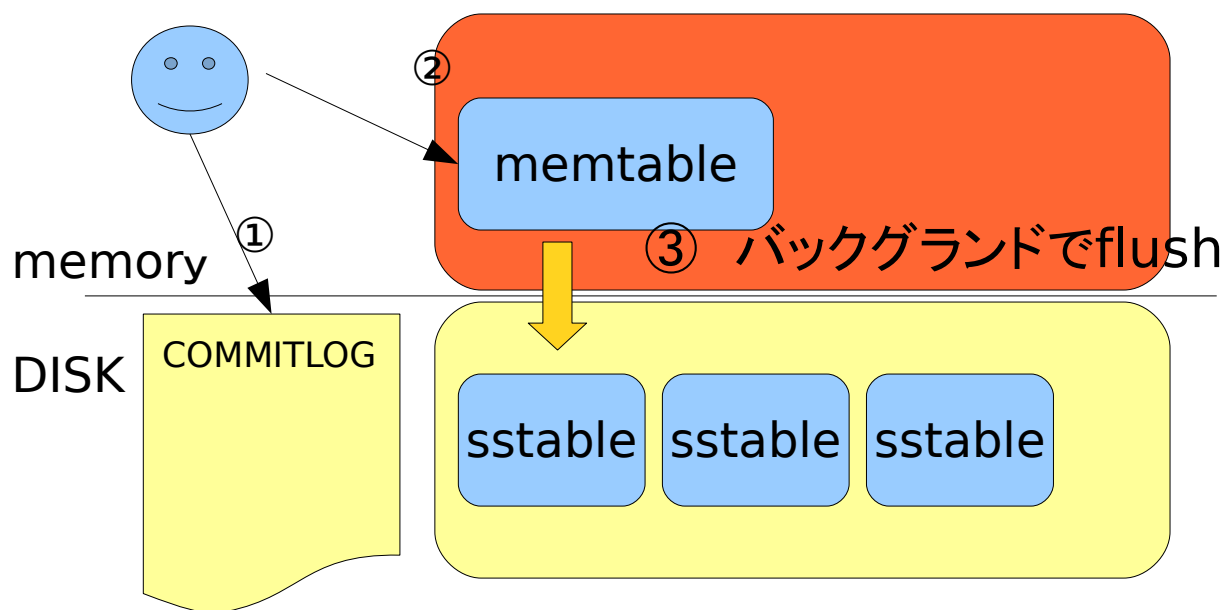
memtableが溢れるとDISKにフラッシュ

sstable ..DISKに書き出されるデータ

(キーでソートされて書き出される)

書き込み処理

- ① 受け取った書き込み要求はCOMMITLOGに書き出される
- ② memtableにも変更内容を反映
ここでユーザに結果を返却
- ③ バックグラウンドでmemtableの内容をフラッシュして、sstableが生成される



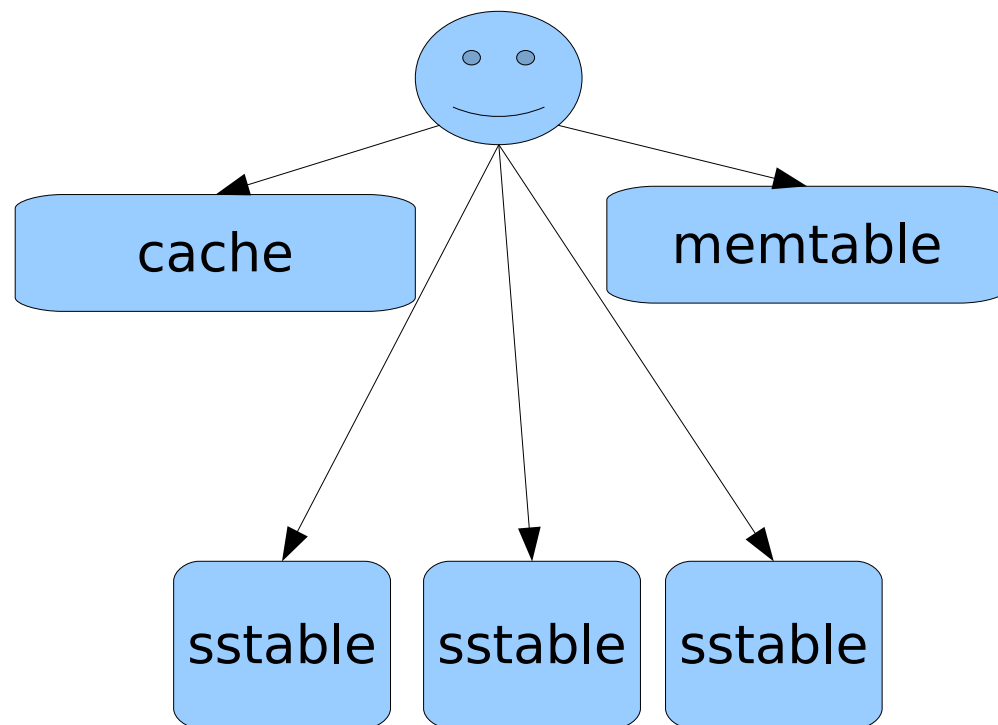
sstableはカラムファミリー単位で管理
memtableをフラッシュするとカラムファミリーにつき一つ作成される

sstableはコンパクションを除き
変更されない

読み込み処理

- メモリ上のmemtableやcacheから探して返事を出す
- メモリ上にデータが無ければsstableから読み込む

複数のsstableをスキャンする必要あり
(indexやbloomfilterで効率化)



障害ケース

- Cassandraのノード障害

ノードが1台でも動いていれば (一応) 動作は可能

クライアントが発行するクエリに依存する

例)

クライアントがConsistency level ALLで書込みを行った場合には、レプリケーション先の全てのノードで書込みを行えない場合にはエラーを返す

Consistencyを優先とすると、Availabilityを損なう

クライアントAPIでは、複数のノード指定が可能なものがあり、障害があったノードを自動で切り離す

例) Java hectorライブラリ

php phpcassaライブラリ

HBase Cassandra 共通事項

コンパクション

- 複数の物理ファイルをまとめる処理

サイズの縮小、リードの効率化

- 有効期限切れのデータやトゥームストーン(削除データ)の削除も同時行われる
- バックグラウンドで自動的に行われ、数個の物理ファイルが一つにまとめられる(マイナーコンパクション)
- 管理コマンドや定期的に一つの物理ファイルにまとめる(メジャーコンパクション)

HbaseもCassandraもランダムアクセスを極力避けるために、直接物理ファイルを変更せずにコンパクションによって、無効データの削除を行い物理ファイルの統合が行われる

このため、WRITEは追記のみであり、極めてWRITE性能が高い

ベンチマーク

- NoSQLベンチマーク比較

<http://www.networkworld.com/news/tech/2012/102212-nosql-263595.html>

A vendor-independent comparison of NoSQL databases

pdfファイル

<http://gotocon.com/dl/goto-cph-2012/slides/big-data/Evaluating%20NoSQL%20performance.pdf>

Cassandra, HBase, MongoDB, Riak, Shared MySQL, MySQLクラスタの性能比較