IIJ INNOVATION INSTITUTE

Ongoing Innovation

# 2010 Techweek
# 分散データベースシステムの比較 (rev1.0)

株式会社 IIJ イノベーションインスティテュート
企画開発センター
久島広幸

# Contents

IIJ INNOVATION INSTITUTE

- データベースに関しての言葉
- 望まれるデータベース
- MongoDB について
- VoltDB について
- まとめ

# データベースに関しての言葉（1）

IIJ INNOVATION INSTITUTE

- ACID 特性（データベースが備えているべき4つの特質）
  - Atomicity (原子性)、Consistensy（一貫性）、Isolation（独立性）、Durabirity（持続性）
  - ISO/IEC 10026-1:1992 Section 4に詳述
- CAP 定理 (Brewer)
  - Consistency (一貫性)、Availability （可用性）、Partition tolerance（分割耐性）を同時に3つ満たす事はできない、という定理
- トランザクション
  - 複数の atomic event をまとめて atomic にすること
  - 結果として排他的
- スキーマ
  - DBの実体のデータ構造：テーブルの定義

# データベースに関しての言葉(2)

IIJ INNOVATION INSTITUTE

- RDBMS
  - スキーマベース
  - SQLによる問い合わせ
- NoSQL
  - No SQL ? Not Only SQL ?
  - BASE　戦略
    - Basically Available, Soft-state, Eventually consistent

# 望まれるデータベース

IIJ INNOVATION INSTITUTE

- ACID 特性を満たす
- CAP 定理を破る
  - C、A、P を同時に満たしたい
  - 可能だろうか
- スキーマからの解放
  - スキーマがすべてを支配。一度決めると融通が利かない
    - テーブルの追加
    - 関係性の追加
- 「神様」のデータベース

# 望まれるデータベースに向けての方法

IIJ INNOVATION INSTITUTE

- ACID 特性
  - 可能な限り維持する（DB であるのだし）
- CAP 定理を破る
  - 種々の工夫を実装し、可能な限り挑む
  - 一般に次のように言われているけれども
    - CA: RDBMS, VoltDB, etc
    - CP: MongoDB, CouchDB, Hbase, etc
    - AP: Cassandra, Dynamo, etc
- スキーマからの解放
  - データ構造に柔軟でありたい
  - 例えば、単純な関係性の採用
    - SQL ではなく、写像を利用
    - key-value の写像関係
- 「神様」のデータベースへの接近をめざす

# 休題

IIJ INNOVATION INSTITUTE

- 「神様」のデータベースへの接近方法について
  - MongoDB
  - VoltDB

について考えてみる。

# MongoDB (1)

IIJ INNOVATION INSTITUTE

- 主な特徴
  - 複数の  key-value 写像をまとめたもの
    - 複数の key-value がまとまったものとしては、古くは termcap, printcap, そして mail header などがある
    - 任意の key に対して index を作成する事ができる
  - スキーマレス
    - 「カラム」というものは存在しない
    - 結果、データ構造が柔軟
  - トランザクション、というものはない
    - ロールバックも不可能
  - 分散化対応
    - replica sets
    - auto-sharding

# MongoDB (2)

IIJ INNOVATION INSTITUTE

## データ形式

| RDBMS | Server | Database | Table | record |
|---|---|---|---|---|
| MongDB | Server | Database | Collection | Document |
| | | | | |

- Document とは？
  - BSON(binary-encoded selialization of JSON-like douments)
  - Max 4MB
  - より大きいサイズのものを扱うには？
    - GridFS 利用（複数のドキュメントに分割）
- Collection とは？
  - 複数の documentを集めたもの
- Database とは？
  - 複数の collection を集めたもの
  - 20000 collections/database

# MongoDB (3)

IIJ INNOVATION INSTITUTE

- プログラミングインタフェース
  - ドライバ関数群として用意
    - C
    - C#
    - C++
    - Java
    - Javascript
    - Perl
    - PHP
    - Python
    - Ruby
  - MongoDB shell も用意

# MongoDB (4)

IIJ INNOVATION INSTITUTE

- Python での例
  - Database := ‘test’
  - Collection := ‘col’

```
>>> con = Connection(“messa1.iij-ii.co.jp”, 27318)
>>> db = con[‘test’]
>>> col = db[‘col’]
>>> col.save(‘service’ : ‘pop3’,  ‘from’: ‘hisasima’)
        …..
>>> for cursor in col.find():
        print cursor
>>> for data in col.find({‘service’:”pop3”})
        print data
```

# MongoDB (5)

IIJ INNOVATION INSTITUTE

- Replica set
  - Master: 1 node
  - Slave: n node (max 6)
  - 特徴
    - master に対してしか save できない
    - master 以外の node に普通に接続すると、その接続は master に振り向けられる（ドライバ関数による仕事）
    - 明示的に指定すれば、slave にも接続できる
    - master の決定は自動（vote 方式）

# MongoDB (5')

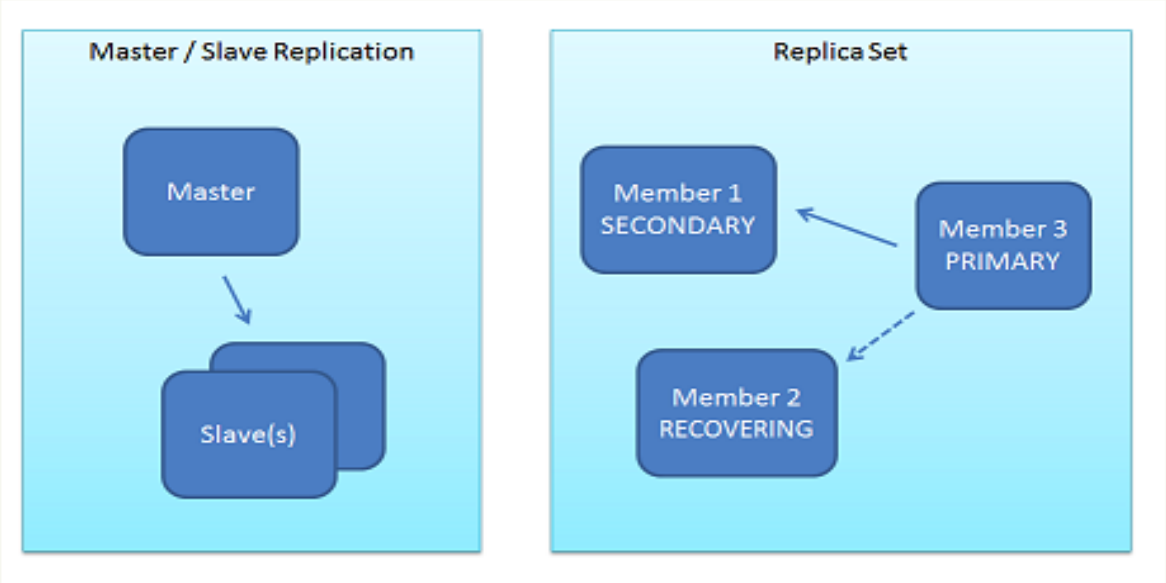
IIJ INNOVATION INSTITUTE

http://www.mongodb.org/display/DOCS/Replication

## Replication

MongoDB supports asynchronous replication of data between servers for failover and redundancy. Only one server (in the set/shard) is active for writes (the primary, or master) at a given time. With a single active master at any point in time, strong consistency semantics are available. One can optionally send read operations to the slaves/secondaries when eventual consistency semantics are acceptable.

- Master-Slave Replication
- Replica Sets



# MongoDB (6)

IIJ INNOVATION INSTITUTE

- Replica set の組み方
  - Node: messa[1-4].iij-ii.co.jp の 4 台
  - 各 node で db directory を作成し mongod を replica set で起動

```
% mkdir /tmp/db
% mongod --port 27318  --replSet tmp --dbpath /tmp/db
```

  - messa1.iij-ii.co.jp だけで mongo シェルを起動し replSet の設定を入れる

# MongoDB (7)

IIJ INNOVATION INSTITUTE

```
messa1% mongo --port 27318
MongoDB shell version: 1.6.0
connecting to: 127.0.0.1:27318/test
> use admin
switched to db admin
> config = {_id: 'tmp', members: [{_id: 0, host: 'messa1.iij-ii.co.jp:27318'},
                                  {_id: 1, host: 'messa2.iij-ii.co.jp:27318'},
                                  {_id: 2, host: 'messa3.iij-ii.co.jp:27318'},
                                  {_id: 3, host: 'messa4.iij-ii.co.jp:27318'}]}
{"_id" : "tmp",
"members" : [
        {"_id" : 0, "host" : "messa1.iij-ii.co.jp:27318"},
        {"_id" : 1, "host" : "messa2.iij-ii.co.jp:27318"},
        {"_id" : 2, "host" : "messa3.iij-ii.co.jp:27318"},
        {"_id" : 3, "host" : "messa4.iij-ii.co.jp:27318”}
]}
> rs.initiate(config)
{"info" : "Config now saved locally.  Should come online in about a minute.","ok" : 1}
```

# MongoDB (8)

IIJ INNOVATION INSTITUTE

しばらく待つと、この設定が messa[2-4].iij-ii.co.jp に伝わり replSet が完成する。さらに、vote の結果、messa1.iij-ii.co.jp が master なる。

```
[ messa1.iij-ii.co.jp]
> rs.isMaster()
{
"ismaster" : true,
"secondary" : false,
"hosts" : [
        "messa1.iij-ii.co.jp:27318”,
        "messa4.iij-ii.co.jp:27318",
        ”messa3.iij-ii.co.jp:27318",
        "messa2.iij-ii.co.jp:27318”],
        "ok" : 1
}
```

# MongoDB (9)

IIJ INNOVATION INSTITUTE

```
[messa2.iij-ii.co.jp]
> rs.isMaster()
{
    "ismaster" : false,
    "secondary" : true,
    "hosts" : [
            "messa2.iij-ii.co.jp:27318",
            "messa4.iij-ii.co.jp:27318",
            "messa3.iij-ii.co.jp:27318",
            "messa1.iij-ii.co.jp:27318"],
    "primary" : "messa1.iij-ii.co.jp:27318",
    "ok" : 1
}
```

# MongoDB (10)

IIJ INNOVATION INSTITUTE

## master 、slave への save

```
Client % mongo messa1.iij-ii.co.jp:27318
MongoDB shell version: 1.6.0
connecting to: messa1.iij-ii.co.jp:27318/test
> db.col.save({'email' : 'hisasima@iij-ii.co.jp'})
> db.col.find()
{
 "_id" : ObjectId("4cbfa7352a7b565bfe8b16d4"), "email" : "hisasima@iij-ii.co.jp"
}
> bye

Client % mongo messa2.iij-ii.co.jp:27318
MongoDB shell version: 1.6.0
connecting to: messa2.iij-ii.co.jp:27318/test
> db.col.save({'email' :  'inoue@iij-ii.co.jp'})
not master
```

# MongoDB (11)

IIJ INNOVATION INSTITUTE

## slave への接続

```
% python
>>> from pymongo import Connection
>>> con = Connection("messa4.iij-ii.co.jp", 27318)
>>> con
Connection(['messa1.iij-ii.co.jp:27318', u'messa3.iij-ii.co.jp:27318', u'messa2.iij-ii.co.jp
27318', u'messa4.iij-ii.co.jp:27318'])

% netstat -a | grep ':27318'
tcp        0      0 dhcp228.iij-ii.co.jp:43501  messa1.iij-ii.co.jp:27318   ESTABLISHED
```

- master である messa1.iij-ii.co.jp に実際に接続

# MongoDB (12)

IIJ INNOVATION INSTITUTE

## slave_okay パラメタの利用

```
>>> con = Connection("messa4.iij-ii.co.jp", 27318, slave_okay = True)
>>> con
Connection(['messa1.iij-ii.co.jp:27318', u'messa3.iij-ii.co.jp:27318', u'messa2.iij-
    ii.co.jp:27318', u'messa4.iij-ii.co.jp:27318'])

% netstat -a | grep ':27318'
tcp        0      0 dhcp228.iij-ii.co.jp:43501  messa1.iij-ii.co.jp:27318   TIME_WAIT
tcp        0      0 dhcp228.iij-ii.co.jp:60754  messa4.iij-ii.co.jp:27318   ESTABLISHED
```

- Slave に接続できる
- 参照のみの場合などは、このパラメタを利用して処理を分散させられる
  - プログラムでの仕事（自動では行ってくれない）

# MongoDB (13)

IIJ INNOVATION INSTITUTE

- master が  down した場合には？
  - 各 node は、「フルメッシュ」で  tcp セッションを張っている
  - 誰が master かは皆が知っている
  - Master が down  した場合には、再度 vote がはじまり、master を決定する

- もちろん、master に  save されたデータは、master が 各 slave に配布

# MongoDB (14)

IIJ INNOVATION INSTITUTE

- Auto sharding
  - sharding の構成要素
    - shard クラスタ (mongod –shardsvr)
      - 実際のデータを保持
    - config サーバ (mongod --configsvr)
      - shard クラスタのメタデータを保持
    - ルータ (mongos)
      - クライアントに対して単一のエンドポイントを提供
      - --configdb パラメータで config サーバを指定
  - 上2つは mongod を replica set にして冗長化することが可能
  - mongos は, mongos 間での調停などないので,「横に並べて」冗長性を確保するしかない

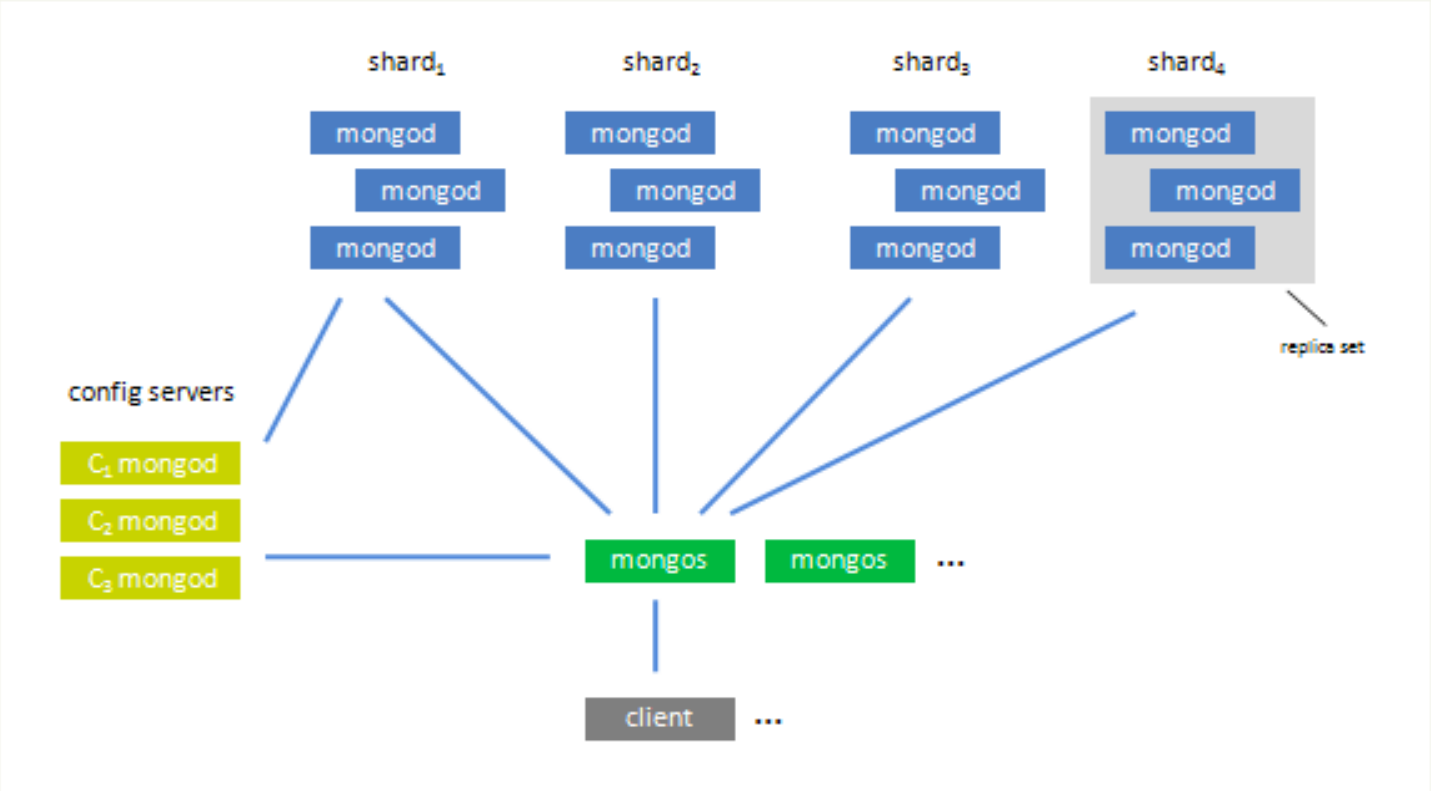
# MongoDB (14')

IIJ INNOVATION INSTITUTE

- http://www.mongodb.org/display/DOCS/Sharding+Introduction

## Architectural Overview

A MongoDB shard cluster consists of two or more shards, one or more config servers, and any number of routing processes to which the application servers connect. Each of these components is described below in detail.



# MongoDB (15)

IIJ INNOVATION INSTITUTE

## パフォーマンスについて

- ドキュメント生成
  - シーケンシャル
    - 以下のようなメール風のデータ構造をしたデータを n 個だけあらかじめ作成し，ループで繰り返しながらサーバに追加していく

```
msg = {
  'id' : 0 ～ (n - 1)
  'service': "tii.service.pop3",
  'from': ”hisasima@iij-ii.co.jp",
  'date': "Tue, 08 Dec 2009 19:57:31 +0900 (JST)",
  'subject': "test",
 'data’:"##########################################################¥n"*20,    }
```

  - バルク
    - 上記データを、一括追加するインタフェースを用いて測定（n 個を一括）
  - 測定はひとつの n につき 3 回ずつ行い，追加処理に要した時間の平均を結果とした.

# MongoDB (16)

IIJ INNOVATION INSTITUTE

## パフォーマンスについて（続き）

- ドキュメント検索
  - あらかじめ 100000 データを保持したデータベースから n 個のドキュメントを検索するのに必要な時間を測定した. 検索対象は, 0 ～ 99999 の乱数を n 個発生しそれらを id として持つドキュメントとした

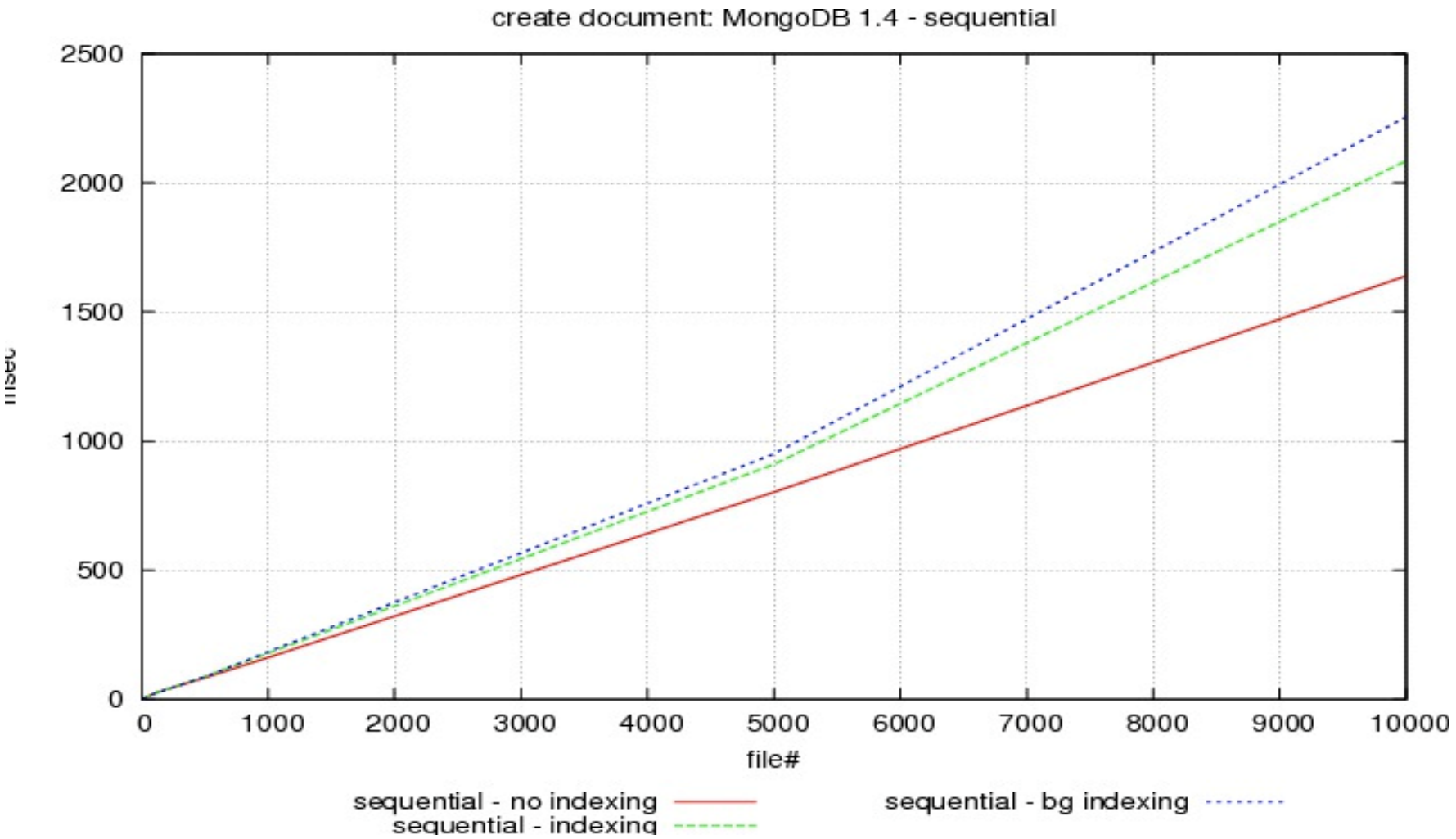
環境

- CentOS 5.3
- 256 MB メモリ
- 32GB disk
- Master のみ
- Vmware ESXi4.0 上のゲスト OS

# MongoDB (17)

IIJ INNOVATION INSTITUTE

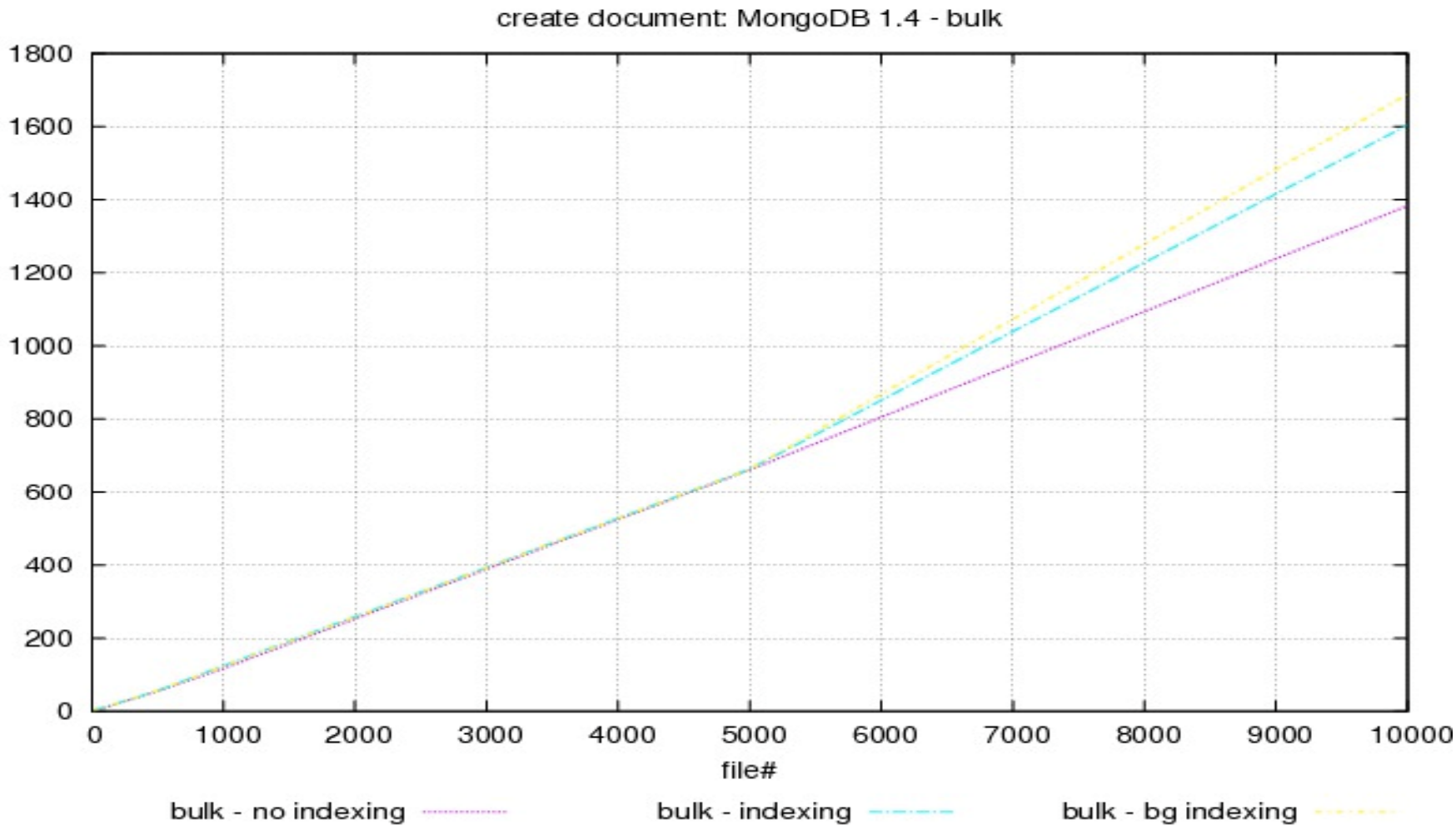
- ドキュメント生成時間 - sequential



# MongoDB (18)

IIJ INNOVATION INSTITUTE

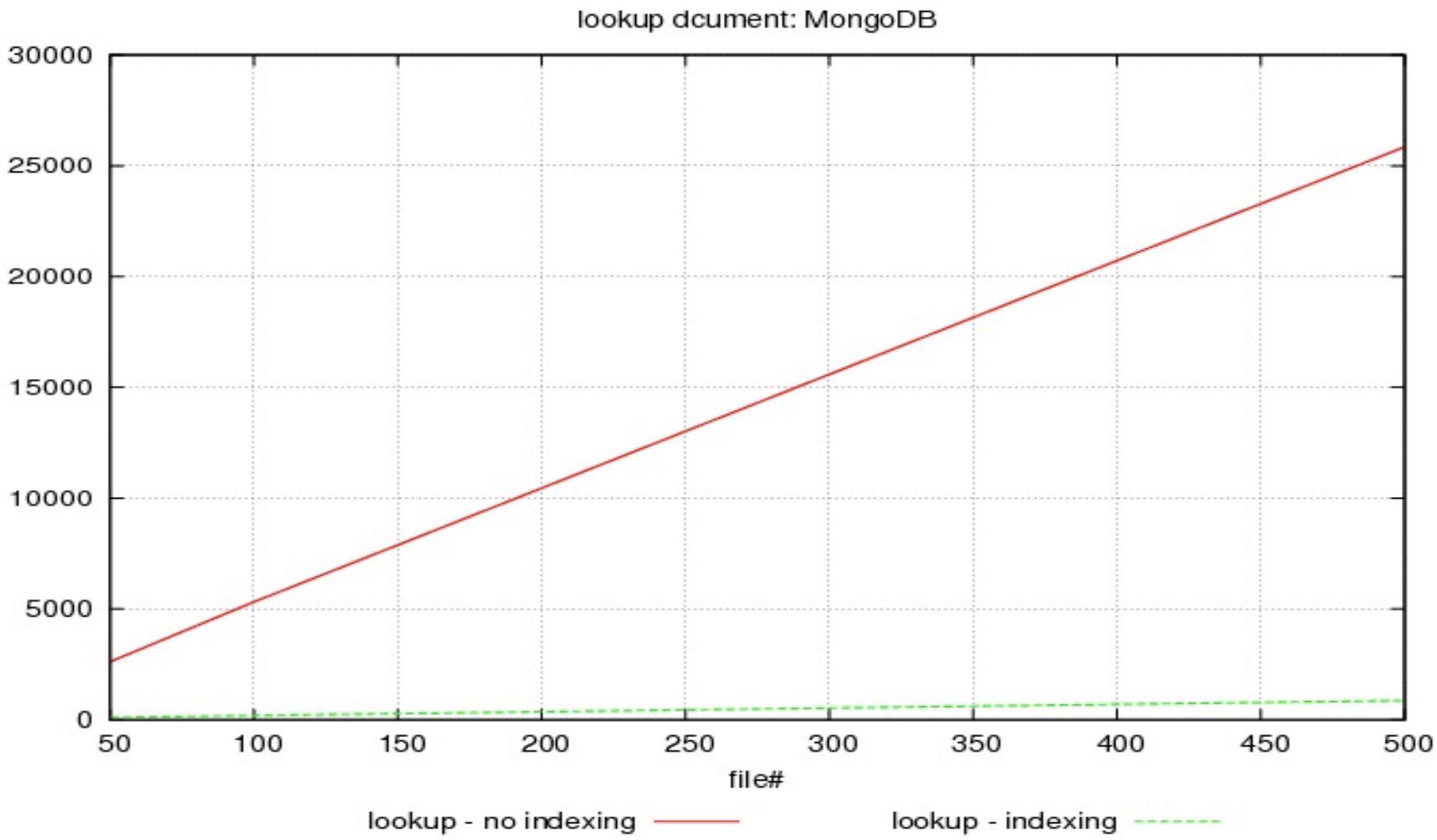
- ドキュメント生成時間 –bulk



# MongoDB (19)

IIJ INNOVATION INSTITUTE

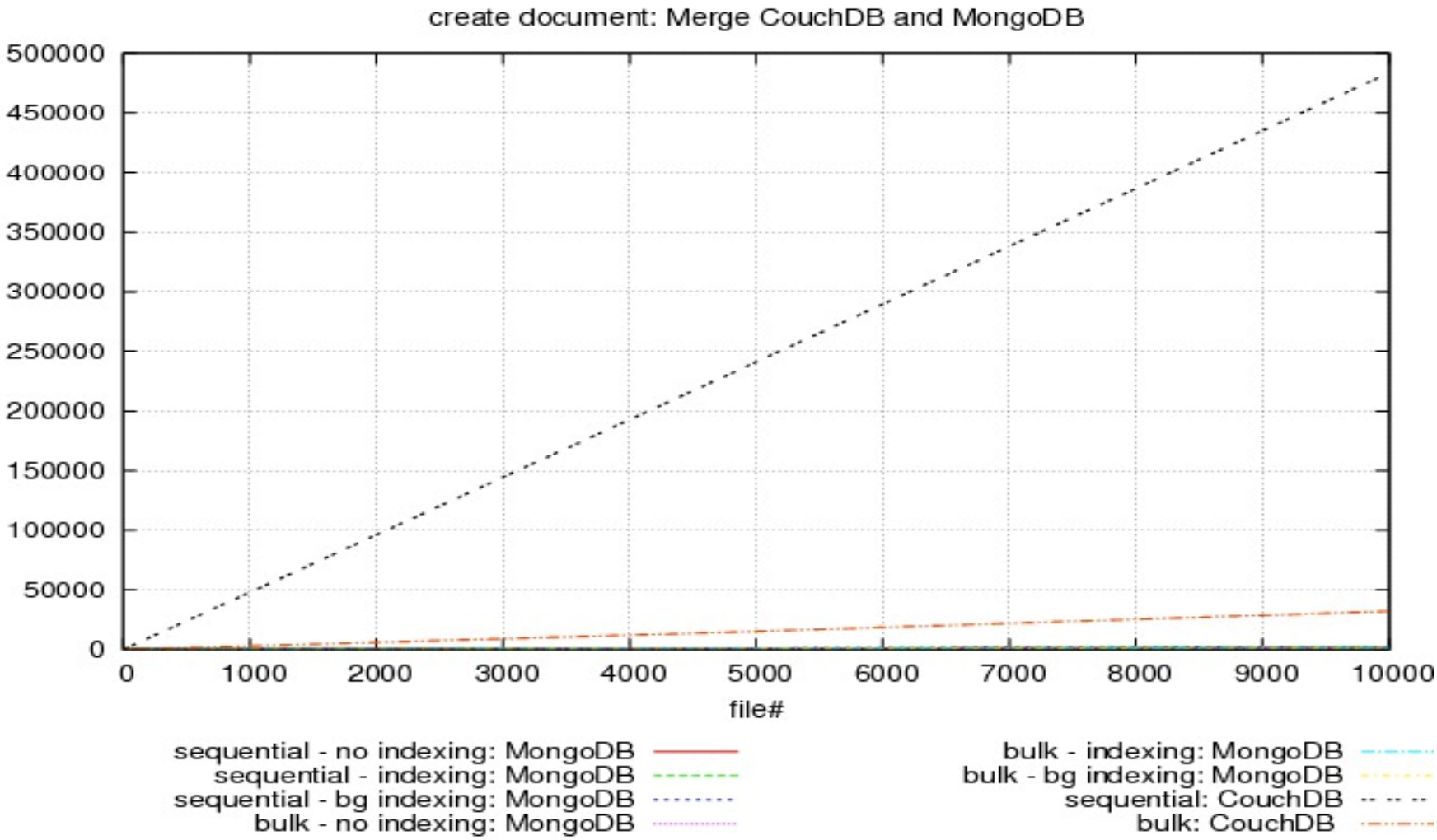
- ドキュメント検索時間



# MongoDB (20)

IIJ INNOVATION INSTITUTE

- couchDB との比較（ドキュメント生成時間）



# VoltDB (1)

IIJ INNOVATION INSTITUTE

- 主な特徴
  - RDBMS である
    - SQL インタフェース
    - スキーマ必要
    - トランザクション対応
  - すべてメモリ上に格納
    - Disk I/O を排除
  - Ad hoc な query は考慮しないという姿勢
    - query パターンをあらかじめ定義
    - stored procedure の集合として振る舞う
  - 複数のサーバで協調して動作
  - すべて  java にて実装

# VoltDB (2)

IIJ INNOVATION INSTITUTE

- 目指しているもの
  - 高速で、ACID を満たす RDBMS
- 方法論
  - すべてメモリ上に格納する
    - Disk I/O を排除、I/O バッファ管理不要
    - ロギングも不要
    - マシンが落ちたら？
      - レプリケーションされている他のマシンのメモリ中には存在するのでかまわない
  - サーバー群は皆対等
    - レプリケーションで一貫性維持、冗長化
  - stored procedure の集合
    - トランザクションをすべて stored procedure 化

# VoltDB (3)

IIJ INNOVATION INSTITUTE

- どのように使うか
  - プロジェクトを定義（xml)
  - スキーマ作成（sql）
  - Stored procedure 作成（java）
  - クライアントプログラム作成（java）
  - カタログ作成（jar）

# VoltDB (4)

IIJ INNOVATION INSTITUTE

- プロジェクトの定義
  - スキーマ定義ファイル名、ストアドプロシージャ名、データベース分割 (パーティション) などの設定

```
<!-- project.xml -->
<?xml version="1.0"?>
<project>
  <database nme="database">
    <schemas>
      <schema path="helloworld.sql" />
    </schemas>
    <procedures>
      <procedure class="Insert" />
      <procedure class="Select" />
    </procedures>
    <partitions>
      <partition table="HELLOWORLD" column="DIALECT" />
    </partitions>
  </database>
</project>
```

# VoltDB (5)

IIJ INNOVATION INSTITUTE

- スキーマの作成
  - データベース定義のための SQL ファイル

```
-- helloworld.sql
CREATE TABLE HELLOWORLD (
    HELLO VARCHAR(15),
    WORLD VARCHAR(15),
    DIALECT VARCHAR(15) NOT NULL,
    PRIMARY KEY (DIALECT)
);
```

# VoltDB (6)

IIJ INNOVATION INSTITUTE

- Stored procedure の作成
  - Java コード

```
/* Insert.java */
@ProcInfo(
    partitionInfo = "HELLOWORLD.DIALECT: 2",
    singlePartition = true
)
public class Insert extends VoltProcedure {
    public final SQLStmt sql = new SQLStmt
        ("INSERT INTO HELLOWORLD VALUES (?, ?, ?);");

    public VoltTable[] run(String hello,
                           String world,
                           String language)
        throws VoltAbortException {
        voltQueueSQL(sql, hello, world, language);
        voltExecuteSQL();

        return null;
    }
}
```

# VoltDB (7)

IIJ INNOVATION INSTITUTE

- クライアントプログラムの作成
  - Java コード

```
/* Client.java */
public class Client {
    public static void main(String[] args) throws Exception {
        /* Instantiate a client and connect to the database. */
        org.voltdb.client.Client myApp;
        myApp = ClientFactory.createClient();
        myApp.createConnection("localhost", "program", "password");

        /* Load the database. */
        myApp.callProcedure("Insert", "Hello", "World", "English");
        myApp.callProcedure("Insert", "Bonjour", "Monde", "French");
        myApp.callProcedure("Insert", "Hola", "Mundo", "Spanish");
        myApp.callProcedure("Insert", "Hej", "Verden", "Danish");
        myApp.callProcedure("Insert", "Ciao", "Mondo", "Italian");
```

# VoltDB (8)

IIJ INNOVATION INSTITUTE

```
/* Retrieve the message. */
        final ClientResponse response = myApp.callProcedure("Select",
                                                            "Spanish");
        if (response.getStatus() != ClientResponse.SUCCESS){
            System.err.println(response.getStatusString());
            System.exit(-1);
        }

        final VoltTable results[] = response.getResults();
        if (results.length == 0 || results[0].getRowCount() != 1) {
            System.out.printf("I can't say Hello in that language.¥n");
            System.exit(-1);
        }

        VoltTable resultTable = results[0];
        VoltTableRow row = resultTable.fetchRow(0);
        System.out.printf("%s, %s!¥n", row.getString("hello"),
                          row.getString("world"));
    }
}
```

# VoltDB (9)

IIJ INNOVATION INSTITUTE

- VoltCompiler でカタログ生成
  - データベーススキーマ、ストアドプロシージャ、プロジェクト定義を元に生成
  - Java コードをコンパイルする

```
% javac Insert.java
% javac Select.java
% javac Client.java
```

  - カタログを作成する

```
% java org.voltdb.compiler.VoltCompiler project.xml helloworld.jar
```

# VoltDB (10)

IIJ INNOVATION INSTITUTE

- サーバ起動
  - 設定ファイル　deployment.xml

```
<!-- deployment.xml -->
<?xml version="1.0"?>
<deployment>
<cluster hostcount="1"
        sitesperhost="2"
        leader="localhost"/>
</deployment>
```

  - 起動

```
% java org.voltdb.VoltDB catalog helloworld.jar deployment deployment.xml
```

- クライアント起動

```
% java Client
```

# VoltDB (11)

IIJ INNOVATION INSTITUTE

- Replication
  - 複数のサーバが起動可能
  - フルメッシュでの接続状態
  - データをすべて sync しあう
  - クライアントへの提供
    - 例えば、Ant 利用の場合、build.xml に複数のホストを記述
    - クライアントは、どのサーバを指定してもよい
    - クライアントは複数のサーバを指定する事も可能

# VoltDB (12)

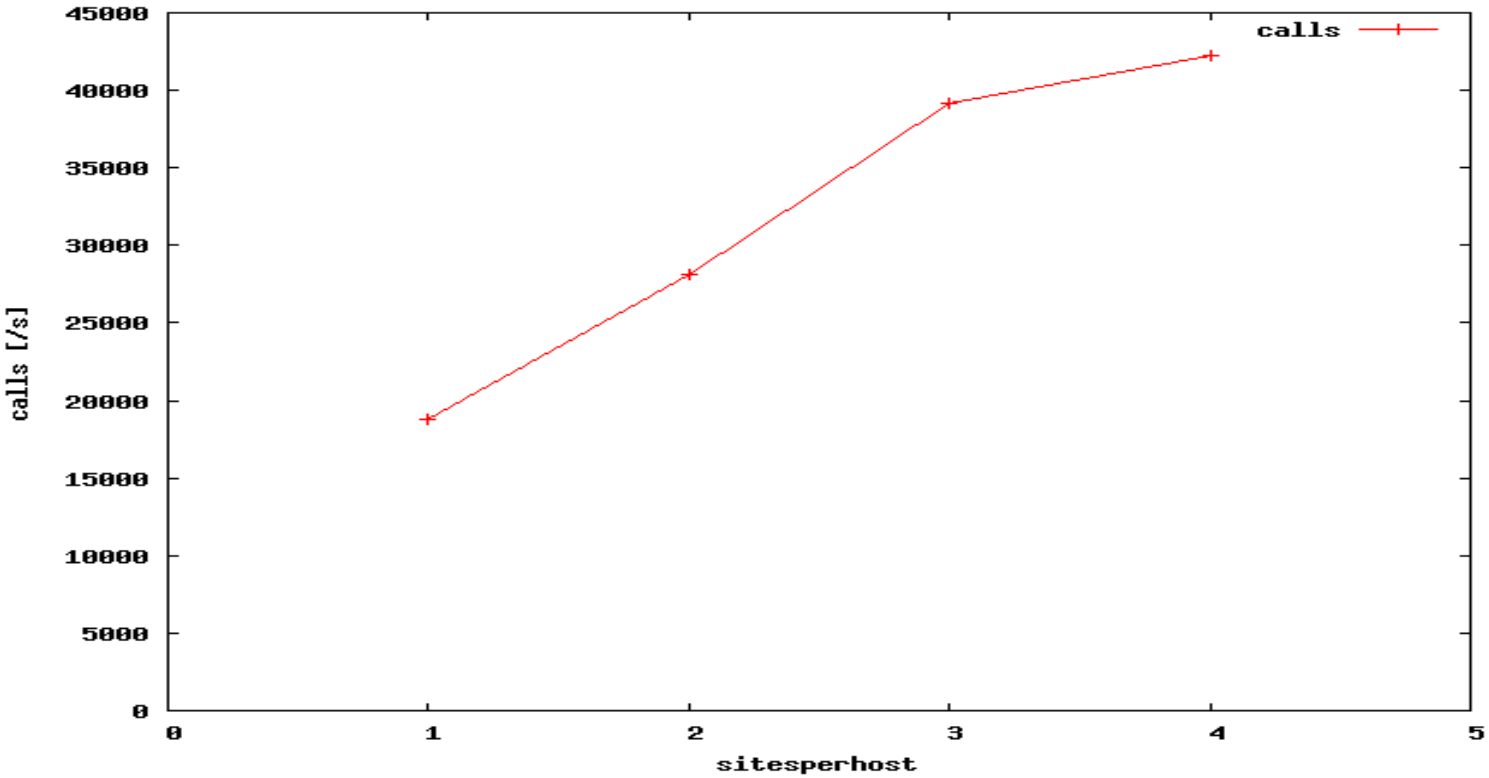
IIJ INNOVATION INSTITUTE

- とある計測
  - VoltDB 付属サンプルコード examples/voter (ネット投票システム) を利用してスループットを測定
  - 実験環境:
    - VMWare ESXi4.0 上の CentOS 5.5 (物理ホスト 1 台につき 1 ゲスト)
    - 1台あたり CPU 4 core, memory 4GB
  - パラメータ
    - ホスト数 (デフォルトは hostcount="1")
    - ホスト毎の site 数 (デフォルトは sitesperhost="2")

# VoltDB (13)

IIJ INNOVATION INSTITUTE

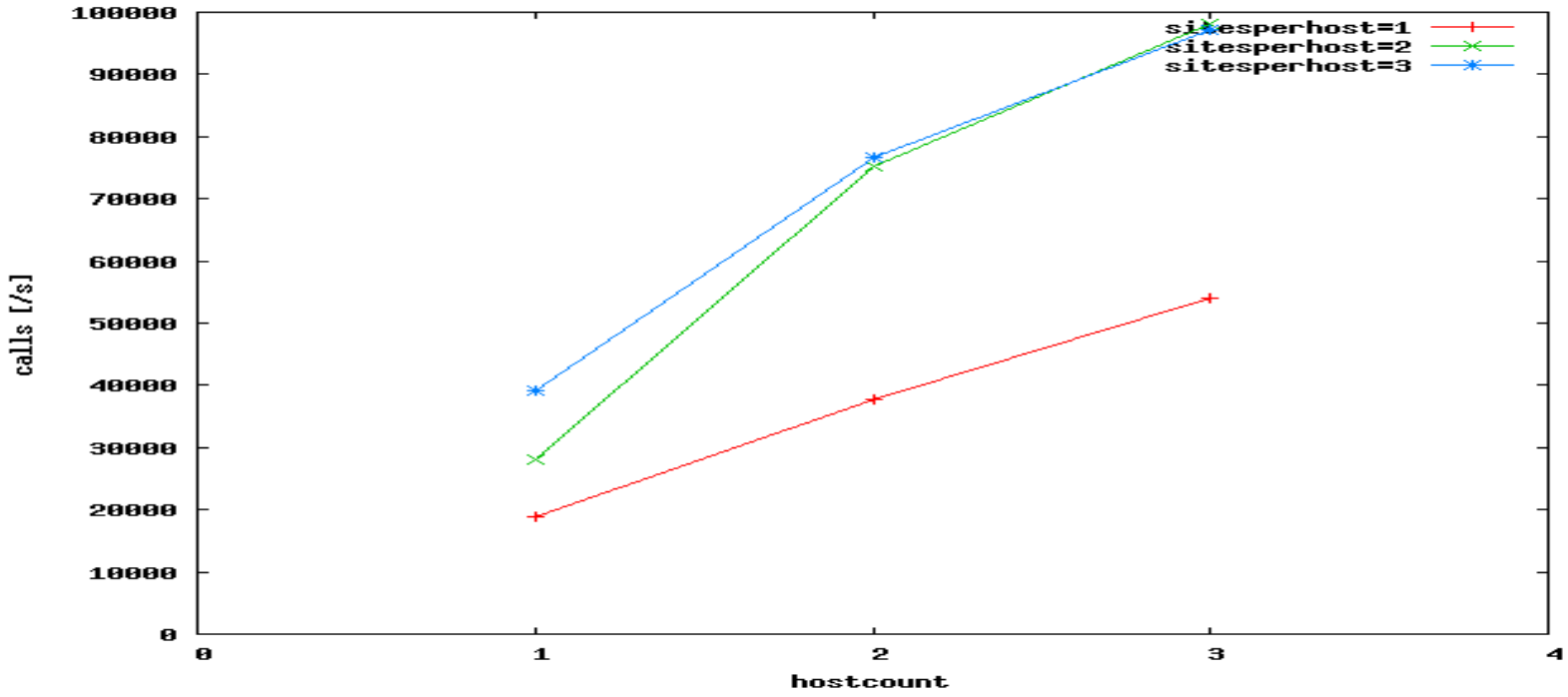
- ホスト数 1 で sitesperhost を変化
  - 1 ～ 3 までは性能が上がっている。4 では上がり方が鈍っている
  - ドキュメントによると core 数の 75% 程度が良いらしい (75% で本来の処理を行い、残りを通信やその他の処理に充てる)



# VoltDB (14)

IIJ INNOVATION INSTITUTE

- sitesperhost 一定で、ホスト数を変化
  - sitesperhost=1
    - ほぼ比例に近く性能が上がっている
  - sitesperhost=2, 3
    - リニアな性能向上が見られない (クライアントの処理速度が足りないせいか？)



# まとめ(1)

IIJ INNOVATION INSTITUTE

- MongoDB
  - DB らしい構造
  - データ構造が柔軟
    - Key-value の集まり
  - 複数の index 作成可能
  - トランザクション定義不可
    - 設計とプログラミングでなんとか回避
    - そもそも対象領域ではない？
  - CouchDB（REST フル（ミドルウエア的））とは対照的
- VoltDB
  - RDBMS である
  - スキーマ必須
  - Query パターンをあらかじめ決めておかねばならない
    - Ad hoc query が使えない事からくる制約
    - すべて stored procedure
  - VoltDB は、互いにレプリケーションするかぎり、各サーバノードでの利用メモリ空間量は同じでなければならないだろう

# まとめ(2)

IIJ INNOVATION INSTITUTE

- MongoDB, VoltDB
  - どちらも動的にノードを追加する方法があるのだろうか？
- DB一般
  - 各データーベースには、得意不得意、向き不向きがある
  - 開発の際にプログラミング言語を選択するがごとく、システム要件に合わせてデータベースを選択する時になってきた
  - データベースの向き不向きにあわせたプログラミングが必要