

MongoDB & Spark

Meteor.js Startup(구 WeBooster) 대표/개발팀장
진정원

소개

- 한국정보통신 SW 연구소
- KTds SW 연구소
- Meteor.js Startup
 - 오픈소스 활성화
 - 정부과제
 - Startup 컨설팅 / 개발지원
 - 카톨릭 대학교 MOU
 - meteorstartup@gmail.com
- 카톨릭대학교 - Javascript & 오픈소스
- 한국인터넷진흥원 - HTML5 / Javascript / CSS 중/고급 코스
- Meteor Startup - Meteor.js, MongoDB 초/중급 코스
- LS산전 - 빅데이터의 이해, 데이터 마이닝과 분석
- Micro Software - 웹앱을 만드는 보다 나은 방법, Meteor.js
- 미림 마에스터 고등학교 - 리눅스/네트워크 이해

MongoDB

DB Rank

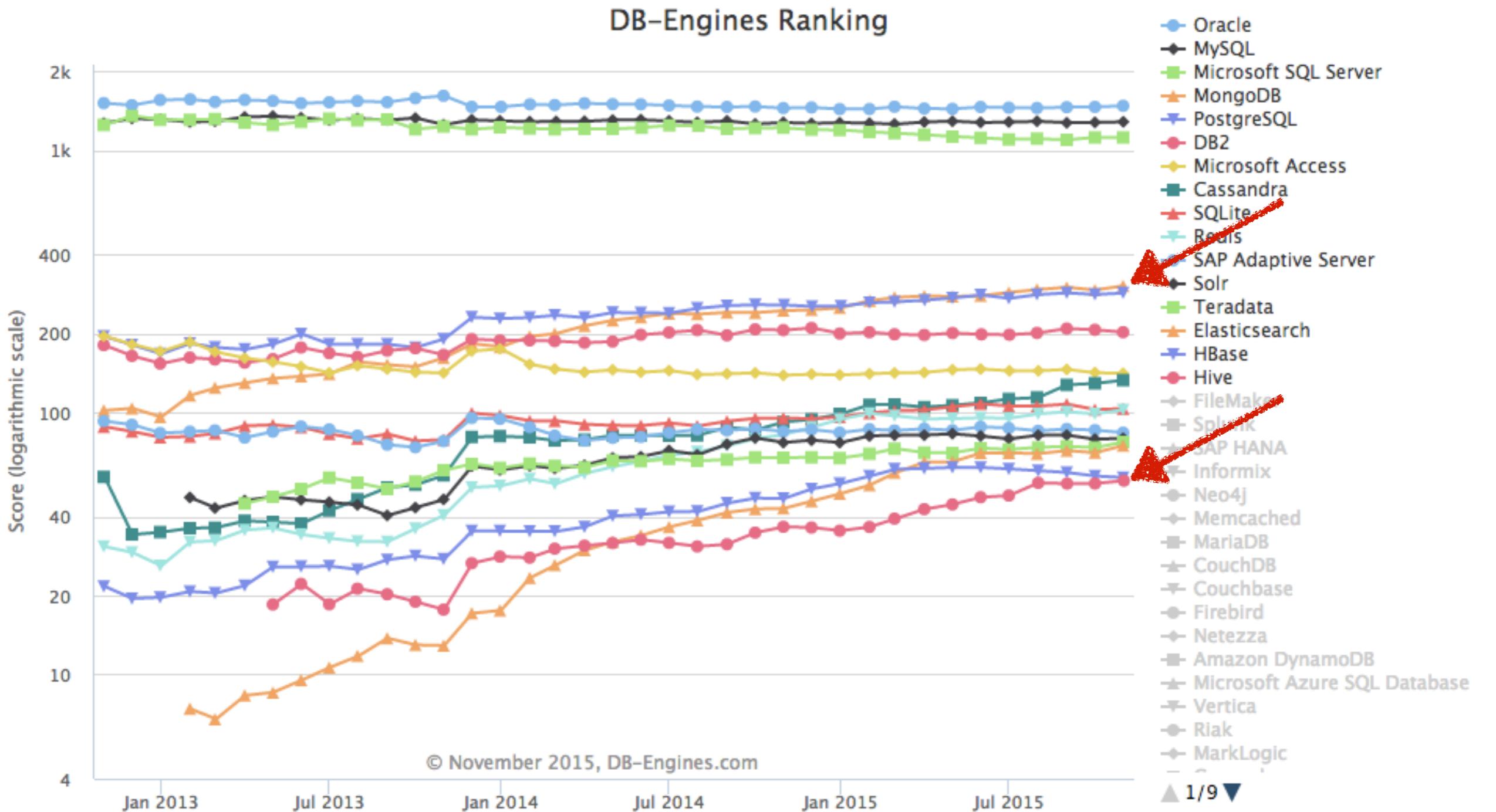
283 systems in ranking, November 2015

Rank			DBMS	Database Model	Score		
Nov 2015	Oct 2015	Nov 2014			Nov 2015	Oct 2015	Nov 2014
1.	1.	1.	Oracle	Relational DBMS	1480.95	+13.99	+28.82
2.	2.	2.	MySQL	Relational DBMS	1286.84	+7.88	+7.77
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1122.33	-0.90	-97.87
4.	4.	↑ 5.	MongoDB +	Document store	304.61	+11.34	+59.87
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	285.69	+3.56	+28.33
6.	6.	6.	DB2	Relational DBMS	202.52	-4.28	-3.71
7.	7.	7.	Microsoft Access	Relational DBMS	140.96	-0.87	+2.12
8.	8.	↑ 9.	Cassandra +	Wide column store	132.92	+3.91	+40.93
9.	9.	↓ 8.	SQLite	Relational DBMS	103.45	+0.78	+8.17
10.	10.	↑ 11.	Redis +	Key-value store	102.41	+3.61	+20.06
11.	11.	↓ 10.	SAP Adaptive Server	Relational DBMS	83.71	-1.93	-0.91
12.	12.	12.	Solr	Search engine	79.77	+0.71	+2.96
13.	13.	13.	Teradata	Relational DBMS	77.08	+3.64	+9.85
14.	14.	↑ 16.	Elasticsearch	Search engine	74.77	+4.55	+31.71
15.	15.	15.	HBase	Wide column store	56.46	-0.78	+9.49
16.	16.	↑ 17.	Hive	Relational DBMS	54.91	+1.35	+18.20
17.	17.	↓ 14.	FileMaker	Relational DBMS	49.99	+1.25	+12.20
18.	18.	↑ 20.	Splunk	Search engine	44.61	+1.11	+12.44

I love Hadoop!

DB Rank

November 2015



MongoDB 활용기업



The New York Times

Telefonica



Genentech

NBCUniversal

NOOKIA

craigslist

Expedia®



eHarmony®



hudl

CARFAX

BROAD
INSTITUTE

Wordnik

- 온라인 Dictionary 사이트
- 3~5TB MySQL to MongoDB 전환
- x20 성능향상



Shutterfly

- 사진 정보 관리 사이트
- SUN Server/JAVA/20TB Oracle DB to MongoDB 전환
- 500% 비용절감
- 900% 성능 향상



Disney

- Disney Interactive Media Group
- Game, Media Content 정보 관리
- MySQL to MongoDB 전환
- Replica Set & Auto Sharding 분산 환경 적용



Forbes

- Business Media Brand
- Oracle to MongoDB 전환
- 전사적으로 전체 데이터에 대해서 확대 적용 중



오라클 정조준한 오픈소스SW 성과공유 사업 모델 선봬

KT그룹 계열사인 KT DS가 오라클 소프트웨어(SW)를 타깃으로 오픈소스 성과 공유형 사업 모델을 선보였다. 유지보수 비용이 높은 오라클 등 외산 SW를 오픈소스SW로 무상 교체해준 후 절감된 비용을 공유하는 사업 구조다. 외산 SW 높은 유지보수 비용에 부담을 느끼는 상당수 기관·기업이 관심을 가질 전망이다.

KT DS는 15일 기자간담회를 갖고 국내 오픈소스SW 시장 활성화와 기업 IT비용 절감을 위해 '오픈소스 성과 공유형 사업모델'을 추진한다고 밝혔다. 오픈소스SW 교체 대상은 운영체계(OS), 데이터베이스관리시스템(DBMS), 웹애플리케이션서버(WAS), 웹서버 등 상용SW다.

KT DS가 중점을 두는 영역은 오라클 DBMS다. 오라클 DBMS 제품은 22%의 높은 유지보수 요율을 요구한다. 최근 오라클 DBMS 도입 기관·기업과 유지보수 요율을 놓고 갈등을 겪었다.

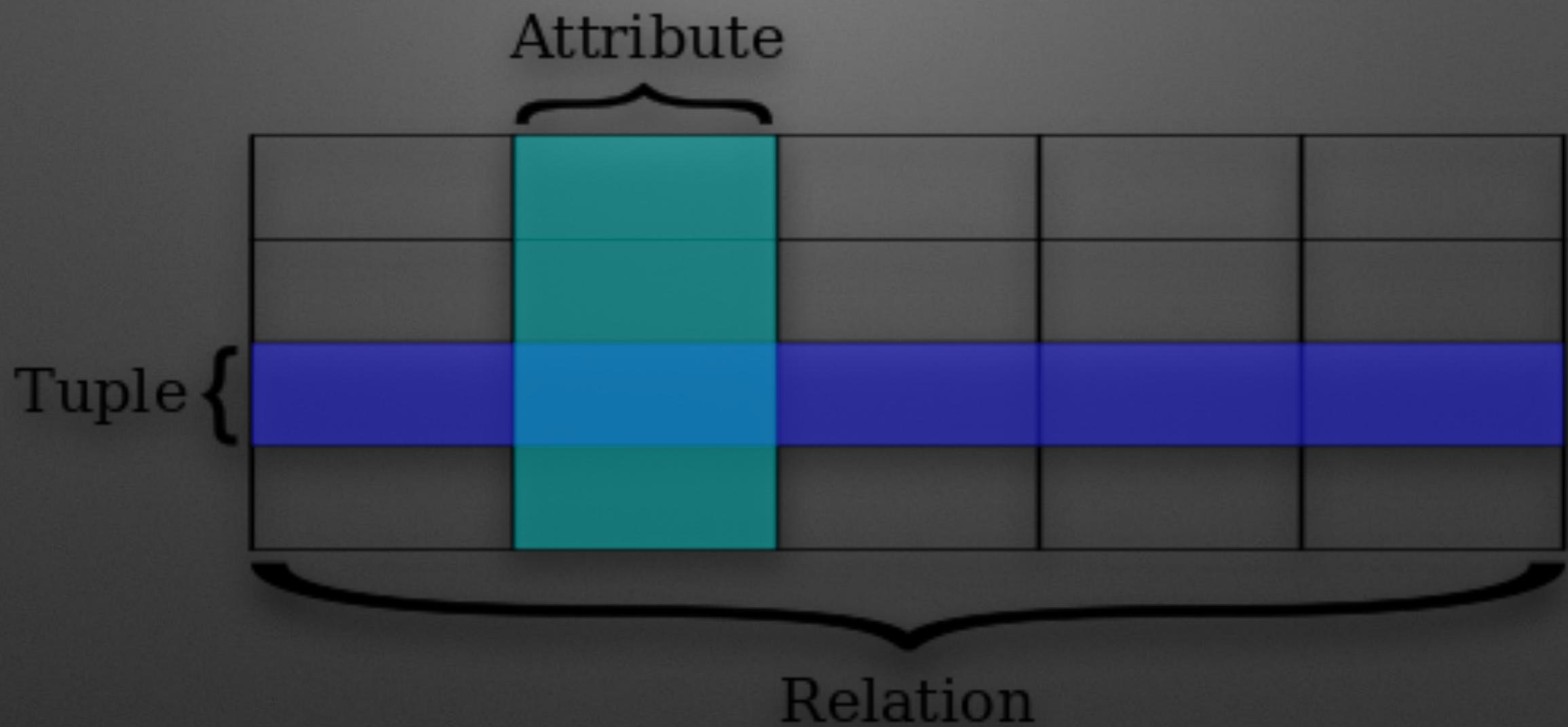
손승혜 KT DS IT서비스혁신센터장은 "아직까지 상당수 기관과 기업이 핵심 영역은 물론이고 비핵심 영역에서도 유지보수 요율이 높은 외산 DBMS에 의존한다"며 "오픈소스SW로 대체하면 비용절감 효과가 크다"고 설명했다.

KT는 신인증시스템 등 주요 30개 정보시스템 DBMS를 오라클 제품 대신 오픈소스인 엔터프라이즈DB(EDB) 포스트그레스 플러스 어드밴스트 서버로 교체했다. 5년간 2000억원 비용절감 효과를 기대한다.

Campung

- 취업 정보 제공 모바일 앱
- JAVA(Spring) + MySQL to Meteor.js + MongoDB
- 월 평균 3000만건의 사용자 로그 데이터 분석
- 어마어마한 성능 향상

RDBMS



RDBMS 빅데이터 대응

- 반정규화 (Denormalization)
- Stored Procedure 사용금지
- 데이터베이스 기능의 사용범위 제한
- 빈번한 쿼리를 미리 만들어 대응
- 보조 색인 사용 금지
- PK 키 외 사용 금지
- 샤딩 (수평적 파티션에 레코드를 논리적으로 분할)

RDBMS 빅데이터 대응

- Replication
 - 복제에 의한 확장
 - Master - Slave 구조
- Partitioning (Sharding)
 - 분할에 의한 확장
 - Join 불가능

비(반)정형 데이터

```
{  
    key: 'value',  
    obj: {  
        key: 'value'  
    },  
    arr: [  
        'val1',  
        'val2'  
    ]  
}
```

<xml>
<some>
 <data>
 value
 </data>
<array>
 <item>
 value
 </item>
</array>
</some>
</xml>

NoSQL DB

KVS	Ordered KVS	Column	Document	Graph
In-Memory	TokyoTyrant	SimpleDB	CouchDB	Neo4j
Memcached	Lightcloud	BigTable	MongoDB	FlockDB
Infinispan	NMDB	Hbase	Lotus Domino	InfiniteGraph
Oracle Coherence	Luxio	Cassandra	Riak	
Disk Based	Memcachedb	HyperTable	ThruDB	
Redis		Azure TS		
Dynamo				
Voldemort				

MongoDB

- 10gen(MongoDB로 사명 변경)에서 만든 **Document** 기반의 NoSQL 스토리지
- **Document-Oriented Storage** : 모든 데이터가 **JSON** 형태로 저장되며 **schema** 없음
- **Full Index Support** : RDBMS에 뒤지지 않는 다양한 인덱싱 제공
- **Replication & High Availability** : 데이터 복제를 통해 가용성을 향상
- **Auto-Sharding** : Primary key를 기반으로 여러 서버에 데이터를 나누는 scale-out 가능

MongoDB

- **Querying** : key 기반의 get, put 뿐만이 아니라 다양한 종류의 쿼리들을 제공
- **Fast In-Place Updates** : 고성능의 atomic **operation**을 지원
- **Map/Reduce** : 맵리듀스를 지원
- **Aggregation**: MongoDB 2.2부터 지원
- **GridFS** : 별도 스토리지 엔진을 통해 파일을 저장
- **BSON**: Document(json)의 직렬 바이너리 포맷

Insert

```
db.inventory.insert(  
  {  
    item: "ABC1",  
    details: {  
      model: "14Q3",  
      manufacturer: "XYZ Company"  
    },  
    stock: [ { size: "S", qty: 25 }, { size: "M", qty: 50 } ],  
    category: "clothing"  
  }  
)
```

Operators

- 비교 연산자
 - \$cmp / \$eq / \$gt / \$gte / \$lt / \$lte / \$ne
- Boolean 연산자
 - \$and / \$not / \$or

Operators

- 산술 연산자
 - \$add / \$devide / \$mod / \$multiply / \$subtract / \$inc
- 문자 연산자
 - \$strcasecmp / \$substr / \$toUpperCase / \$toLowerCase
- 정규표현식
 - db.employees.find({name: {\$regex: 'kim*', \$options: 'm'}}, {empno:", ename:"});

Find

```
db.inventory.find( {} )
db.inventory.find( { type: "snacks" } )
db.inventory.find( { type: { $in: [ 'food', 'snacks' ] } } )
db.inventory.find( { type: 'food', price: { $lt: 9.95 } } )
db.inventory.find(
{
  $or: [ { qty: { $gt: 100 } }, { price: { $lt: 9.95 } } ]
}
)
db.inventory.find( { type: 'food' }, { item: 1, qty: 1 } )
db.inventory.find( { type: 'food' }, { item: 1, qty: 1, _id:0 } )
```

id

- ObjectId is a 12-byte BSON type, constructed using:
 - a 4-byte value representing the seconds since the Unix epoch,
 - a 3-byte machine identifier,
 - a 2-byte process id, and
 - a 3-byte counter, starting with a random value.
- ObjectId("507f191e810c19729de860ea").getTimestamp()
- ISODate("2012-10-17T20:46:22Z")
- ObjectId("507f191e810c19729de860ea").str

Cursor

```
var myCursor = db.inventory.find( { type: 'food' } );
while (myCursor.hasNext()) {
  print(tojson(myCursor.next()));
}
```

```
var myCursor = db.inventory.find( { type: 'food' } );
myCursor.forEach(printjson);
```

Explain

- `db.collection.explain("options")`
- `db.collection.find().explain("options")`
- `options`
 - `queryPlanner`
 - `executionStats`
 - `allPlansExecution`

Update

```
db.inventory.update(  
  { item: "MNO2" },  
  {  
    $set: {  
      category: "apparel",  
      details: { model: "14Q3", manufacturer: "XYZ Company" }  
    },  
    $currentDate: { lastModified: true }  
  },  
  { multi: true }  
)
```

Upsert

```
db.inventory.update(  
  { item: "TBD1" },  
  {  
    item: "TBD1",  
    details: { "model": "14Q4", "manufacturer": "ABC Company" },  
    stock: [ { "size": "S", "qty": 25 } ],  
    category: "houseware"  
,  
  { upsert: true }  
)
```

Save

```
db.products.save( { item: "book", qty: 40 } )  
db.products.save( { _id: 100, item: "water", qty: 30 } )  
db.products.save(  
  { item: "envelopes", qty : 100, type: "Clasp" },  
  { writeConcern: { w: "majority", wtimeout: 5000 } }  
)
```

WriteConcern

- { w: <value>, wtimeout: <number> }
- w options
 - 1(default): Requests acknowledgement that the write operation has propagated to the standalone mongod or the primary in a replica set.
 - 0: Requests no acknowledgment of the write operation.
 - “majority”: Requests acknowledgment that write operations have propagated to the majority of voting nodes [1], including the primary, and have been written to the on-disk journal for these nodes.

Remove

```
db.inventory.remove({})
```

```
db.inventory.remove( { type : "food" } )
```

```
db.inventory.remove( { type : "food" }, 1 )
```

```
db.inventory.remove( { type : "food" }, true )
```

Index

- Index명 대소문자 구분
- Sort()와 Limit()을 통한 성능 향상
- db.collection.ensureIndex();
- db.collection.reIndex();
- db.collection.dropIndexes();

Index

- Non Unique Index: Field의 중복값 허용
 - Single Key Index: {empno: -1}
 - Compound Key Index: {empno: -1, empname: 1}
- Unique Index: Field의 유일값. {unique: true}
- Sparse Index: 출현 빈도가 낮은 Field. {sparse: true}
- Background Index: {background: true}
 - 시스템 자원의 부족 / 대용량 데이터의 경우 Seamless 한 Index 생성 가능

GeoSpatial Index

- 좌표에 의한 2차원 구조의 Collection에 2D Index를 생성
 - {pos: '2d'}
 - 다양한 연산자 제공
 - \$near / \$center / \$box / \$polygon /
\$centerSphere / \$nearSphere

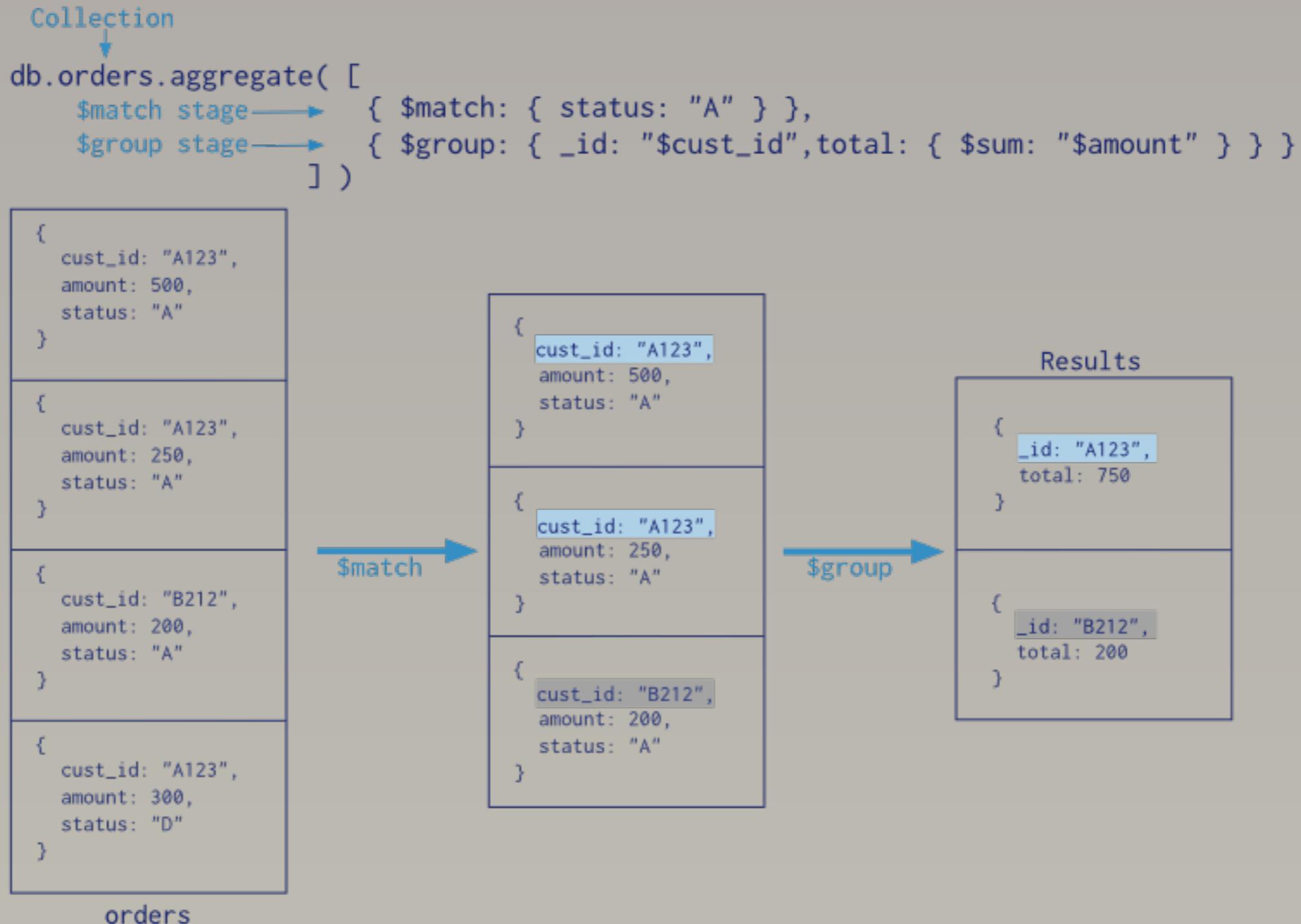
GeoSpatial Index

- **\$near:** 가장 가까운 좌표
- **\$center:** 가장 가까운 원형 좌표
- **\$box:** 가장 가까운 사각형 좌표
- **\$polygon:** 가장 가까운 다면형 좌표
- **\$centerSphere:** 원형내 좌표 검색
- **\$nearSphere:** 원형외 좌표 검색

Aggregation

- 순차적으로 데이터를 실시간으로 추출 및 가공
- Sharding Supported
- MapReduce와 비슷한 동작을 수행하지만 빠른 성능 보장
 - 쉽다!!
 - 빠르다!! Use Memory!!
 - 최적화된 연산자 제공
 - \$project, \$match, \$group, \$sort, \$limit, \$skip
- Runs inside MongoDB local data

Aggregation



SQL VS Aggregation

SQL

```
SELECT COUNT(*) AS count  
FROM order
```

```
SELECT cust_id, SUM(price) AS total_price  
FROM order  
GROUP BY cust_id  
ORDER BY total_price
```

```
SELECT cust_id, count(*)  
FROM order  
GROUP BY cust_id  
HAVING count(*) > 1
```

AGGREGATION

```
db.order.aggregate([  
  $group: {_id: null, count: {$sum: 1}}  
])
```

```
db.order.aggregate([  
  {$group: {_id: "$cust_id", total_price: {$sum:  
    "$price" }}},  
  {$sort: {total_price: 1}}  
])
```

```
db.order.aggregate([  
  {$group: {_id: "$cust_id", count: {$sum:  
    1}}},  
  {$match: {count: {$gt: 1}}}  
])
```

to Dynamic Queries

```
db.logs.find({'path': '/index.html'})
```

```
db.logs.find({'time': {  
    'time': {'$gte': new Date(2013, 0), '$lt': new Date(2013, 0)}  
}})
```

```
db.logs.find({  
    'host': '127.0.0.1',  
    'time': {'$gte': new Date(2013, 0), '$lt': new Date(2013, 1)}  
})
```

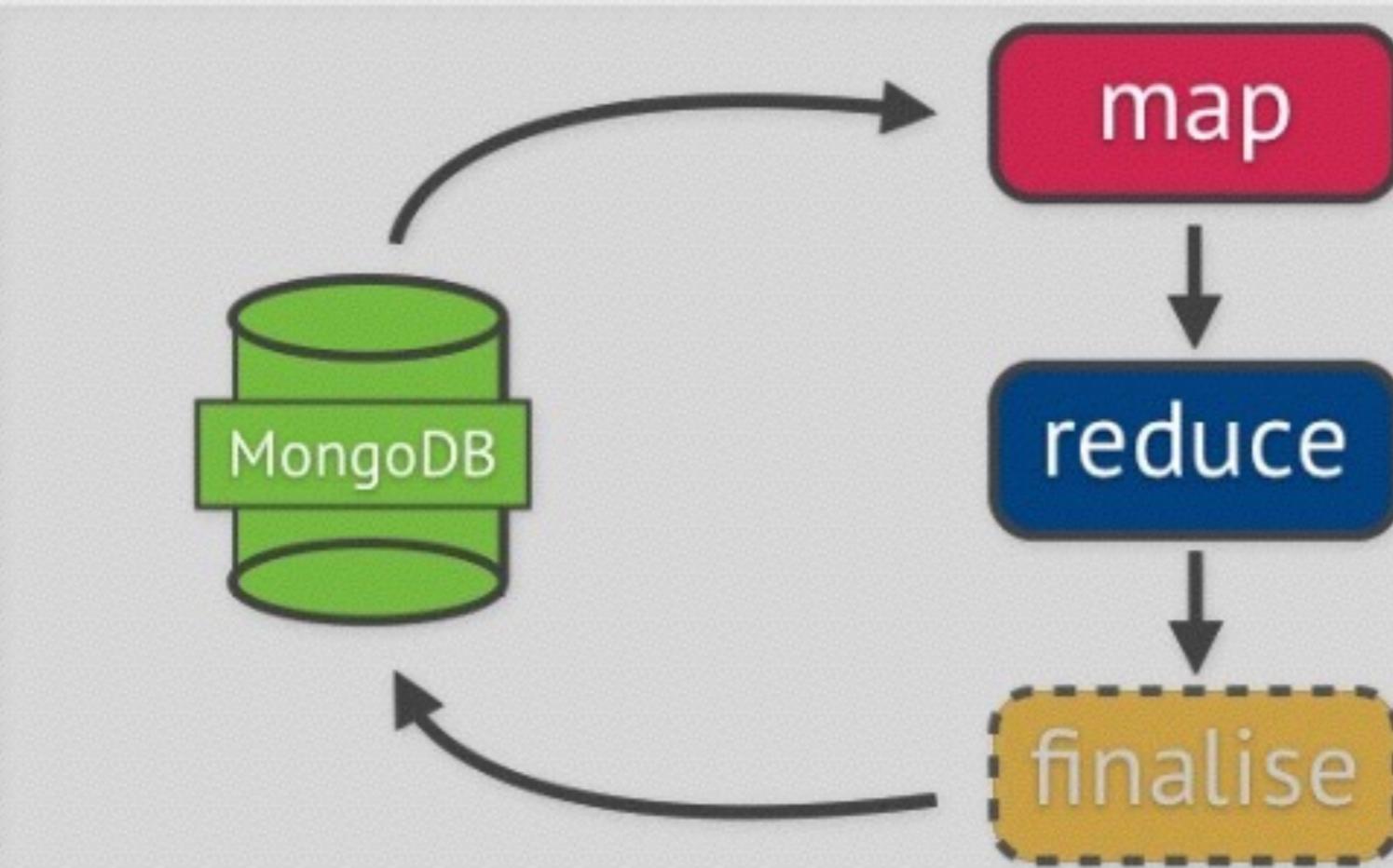
to Aggregation

```
db.logs.aggregate([
  {'$match': {
    'time': {'$gte': new Date(2013, 0), '$lt': new Date(2013, 1)}},
  {'$project': {
    'path': 1,
    'date': {
      'y': {'$year': '$time'},
      'm': {'$month': '$time'},
      'd': {'$dayOfMonth': '$time'}}}},
  {'$group': {
    '_id': {
      'p': '$path',
      'y': '$date.y',
      'm': '$date.m',
      'd': '$date.d'},
      'hits': {'$sum': 1}}}
])
```

Result

```
{  
  'ok': 1,  
  
  'result': [  
  
    {'_id': {'p': '/index.html', 'y': 2013, 'm': 1, 'd': 1}, 'hits': 123},  
  
    {'_id': {'p': '/index.html', 'y': 2013, 'm': 1, 'd': 2}, 'hits': 205},  
  
    {'_id': {'p': '/index.html', 'y': 2013, 'm': 1, 'd': 3}, 'hits': 332},  
  
    {'_id': {'p': '/index.html', 'y': 2013, 'm': 1, 'd': 4}, 'hits': 173},  
  
    {'_id': {'p': '/index.html', 'y': 2013, 'm': 1, 'd': 4}, 'hits': 32},  
  
  ]  
}
```

Map/Reduce



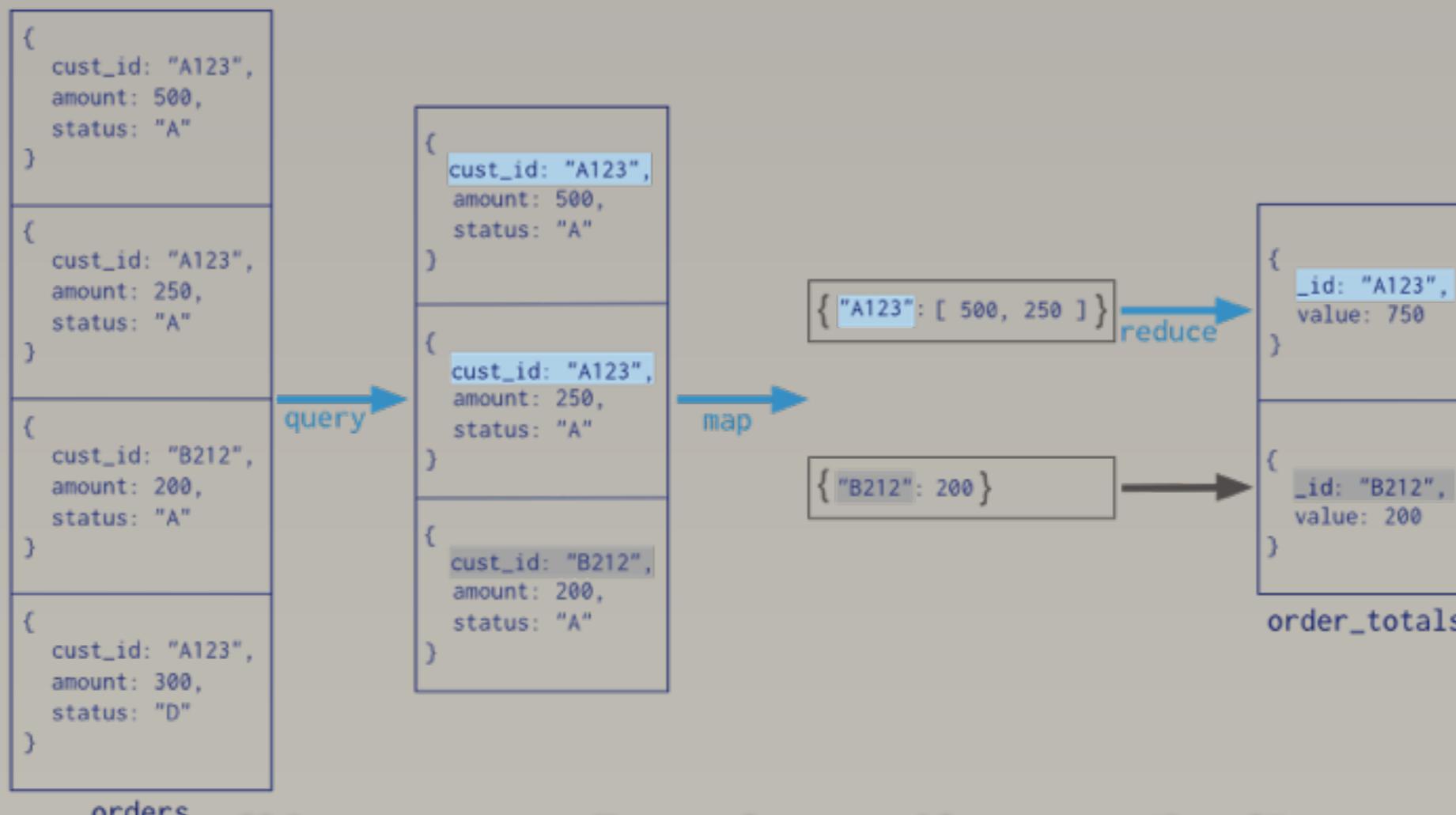
MongoDB Map/Reduce

Map/Reduce

- Hadoop의 Map/Reduce와 유사
- JavaScript
 - pros: 가능성, 표현력 증대. 하둡(JAVA)에 비해 쉽다.
 - cons: Aggregation에 비해 성능 저하
- Runs inside MongoDB local data

Map/Reduce

```
Collection  
↓  
db.orders.mapReduce(  
    map → function() { emit( this.cust_id, this.amount ); },  
    reduce → function(key, values) { return Array.sum( values ) },  
    {  
        query → { status: "A" },  
        output → { out: "order_totals" }  
    }  
)
```



Map/Reduce

```
var map = function() {  
    var key = {  
        p: this.path,  
        d: new Date(  
            this.ts.getFullYear(),  
            this.ts.getMonth(),  
            this.ts.getData(),  
            this.ts.getHours(),  
            0,0,0)};  
    emit(key, {hits: 1});  
};
```

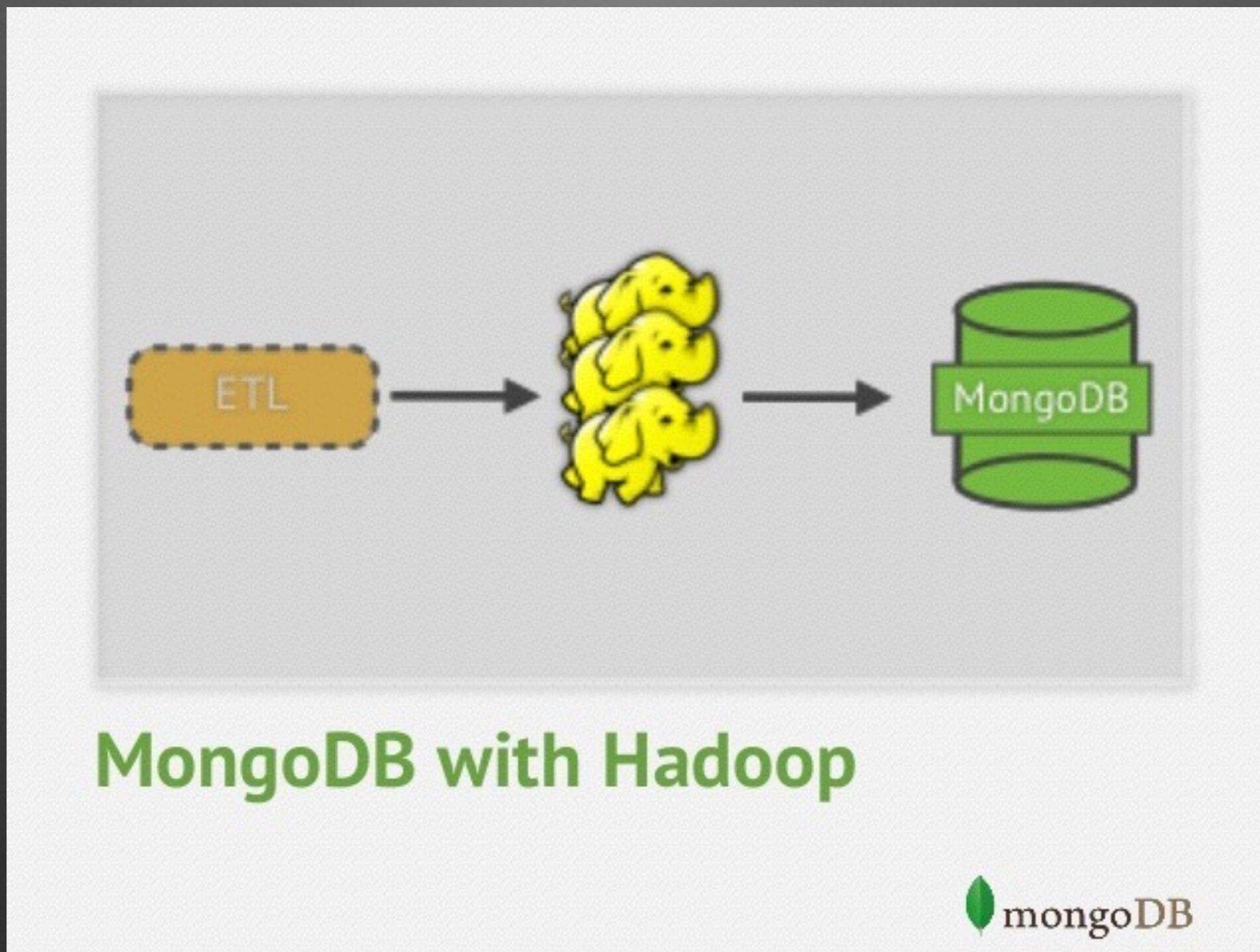
Map/Reduce

```
var reduce = function(key, values) {  
    var r = {hits: 0};  
    values.forEach(function(v) {  
        r.hits += v.hits;  
    });  
    return r;  
};
```

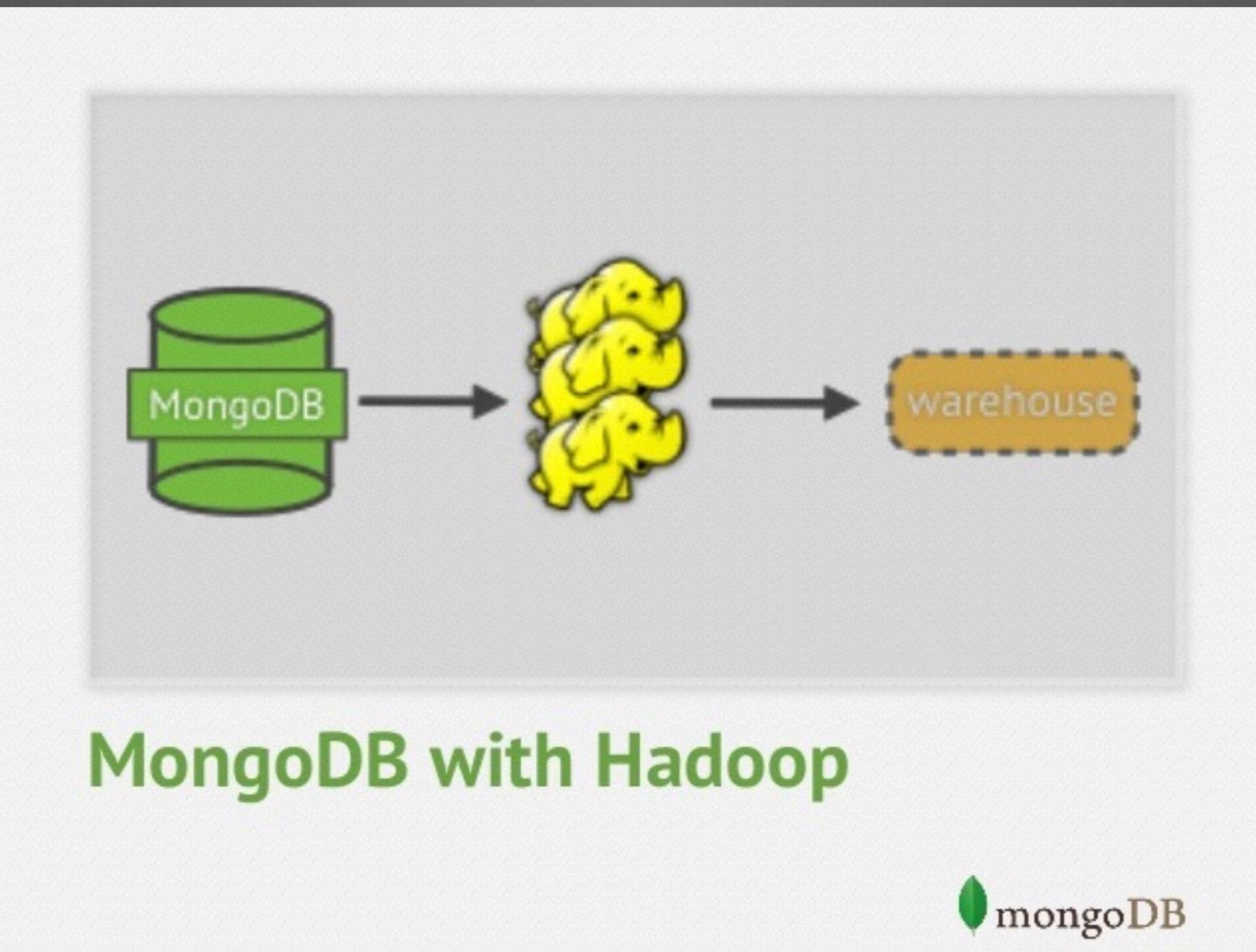
Map/Reduce

```
var query = {  
    'ts': {  
        '$gte': new Date(2013, 0, 1),  
        '$lte': new Date(2013, 0, 31)} };  
  
db.logs.mapReduce(map, reduce, {  
    'query': query,  
    'out': {  
        'reduce': 'stats.monthly' }  
});
```

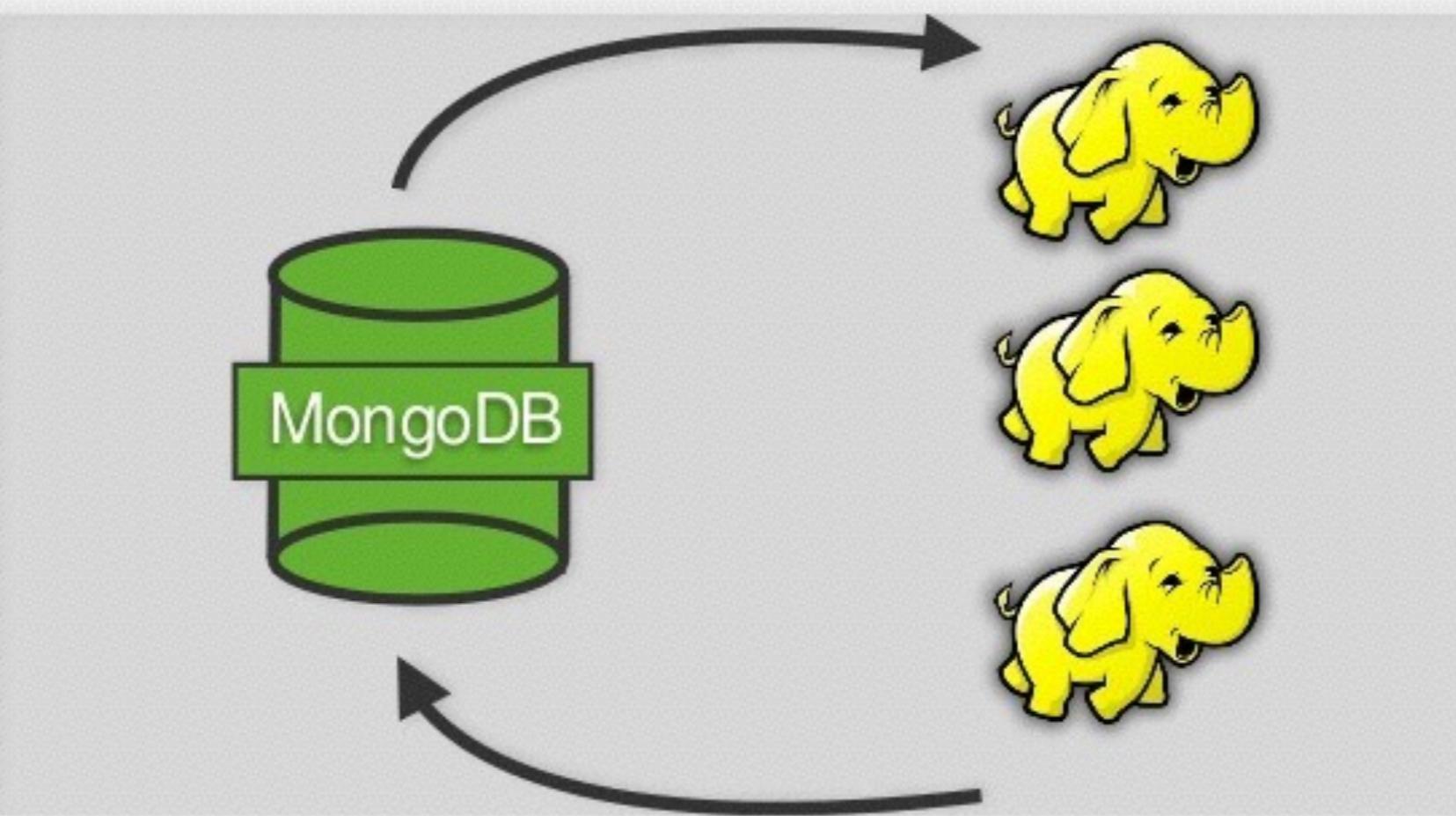
Hadoop Connector



Hadoop Connector



Hadoop Connector



MongoDB with Hadoop

Hadoop Connector

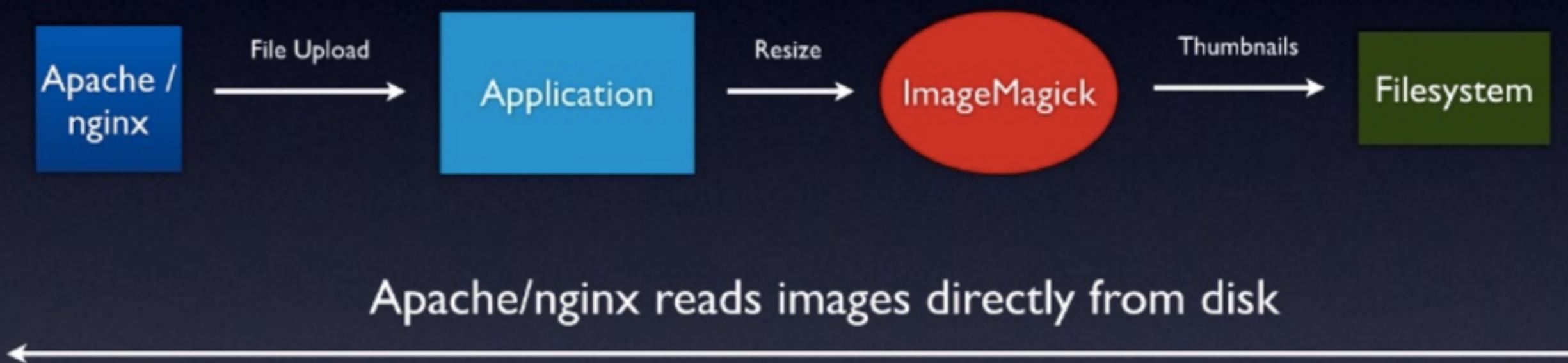
- Adapter가 MongoDB input Collection을 전달받아 data 조각(split)으로 나눔
- 각 조각이 하둡 노드에 할당
- 병렬 하둡 노드가 해당 데이터 조각(혹은 BSON)으로 프로세싱
- 연산 결과(output)을 MongoDB의 Collection에 저장

Hadoop Connector

- FULL multi-core 병렬 연산
- FULL integration with Hadoop ecosystem
 - Pig, Hive...
- Amazon Elastic MapReduce && S3 || HDFS

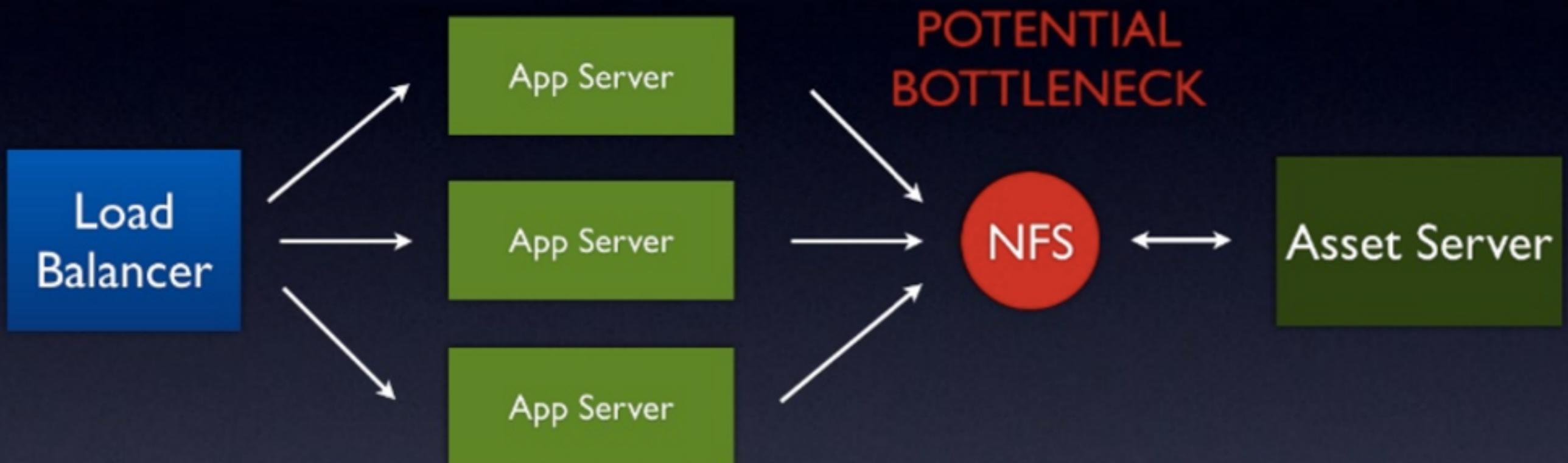
GridFS

Single Server Assets



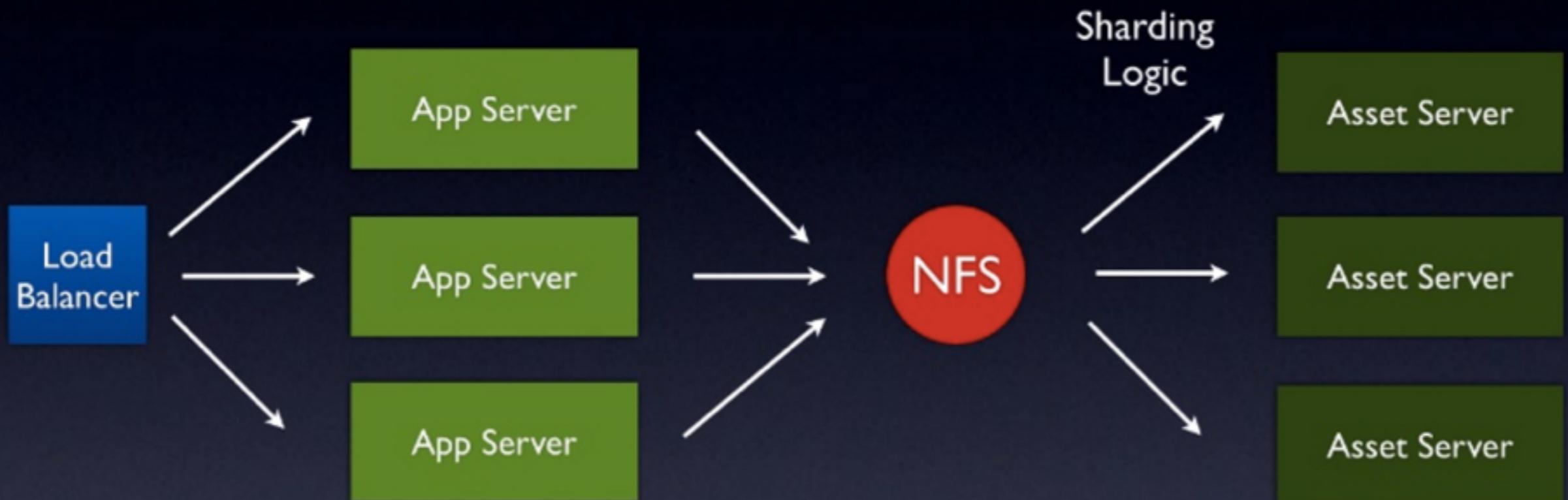
GridFS

Multi-Server Assets



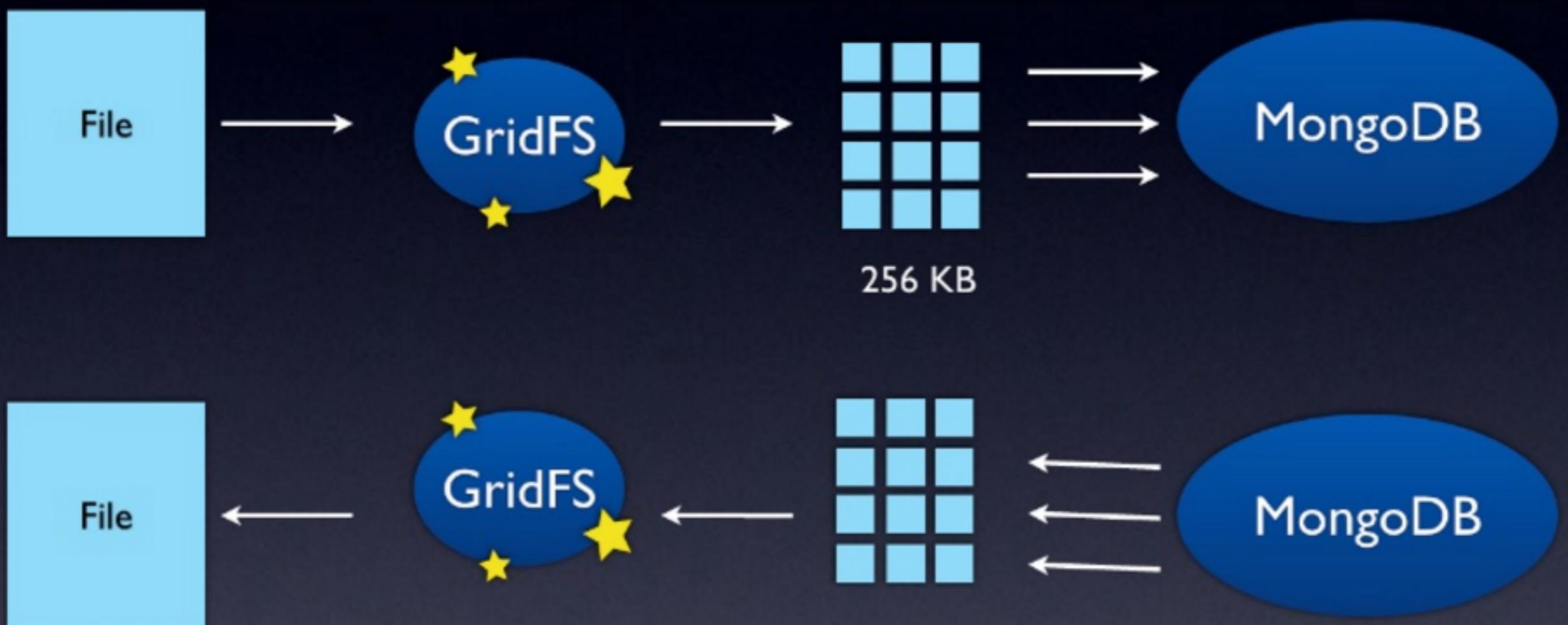
GridFS

Shareded Assets



GridFS

So, GridFS



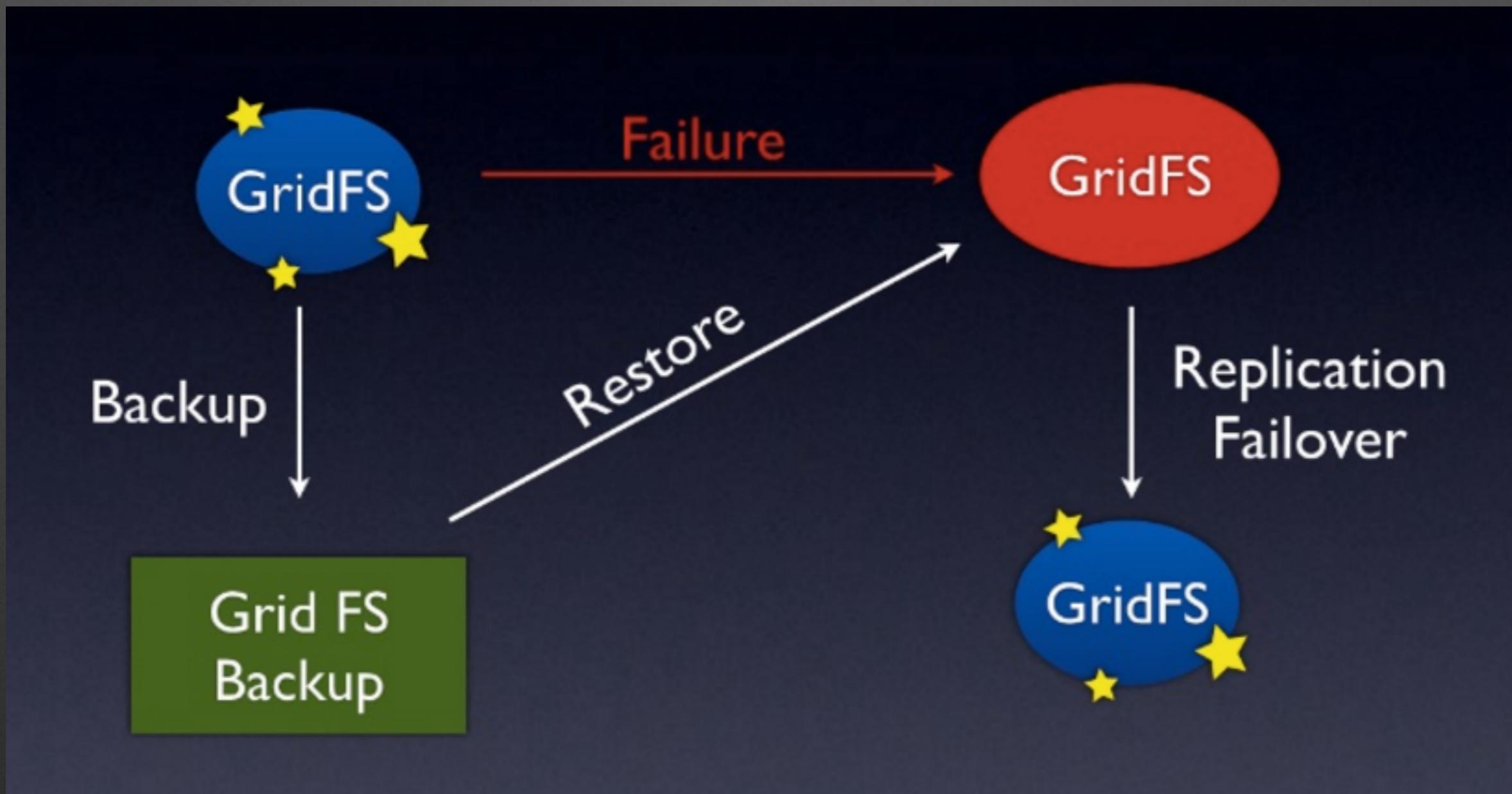
GridFS

Chunk Failover



GridFS

Chunk Recover



When to use GridFS

1. Storing user generated file content
2. Accessing portions of file content
3. Storing documents greater than 16MB in MongoDB
4. Overcoming file system limitations

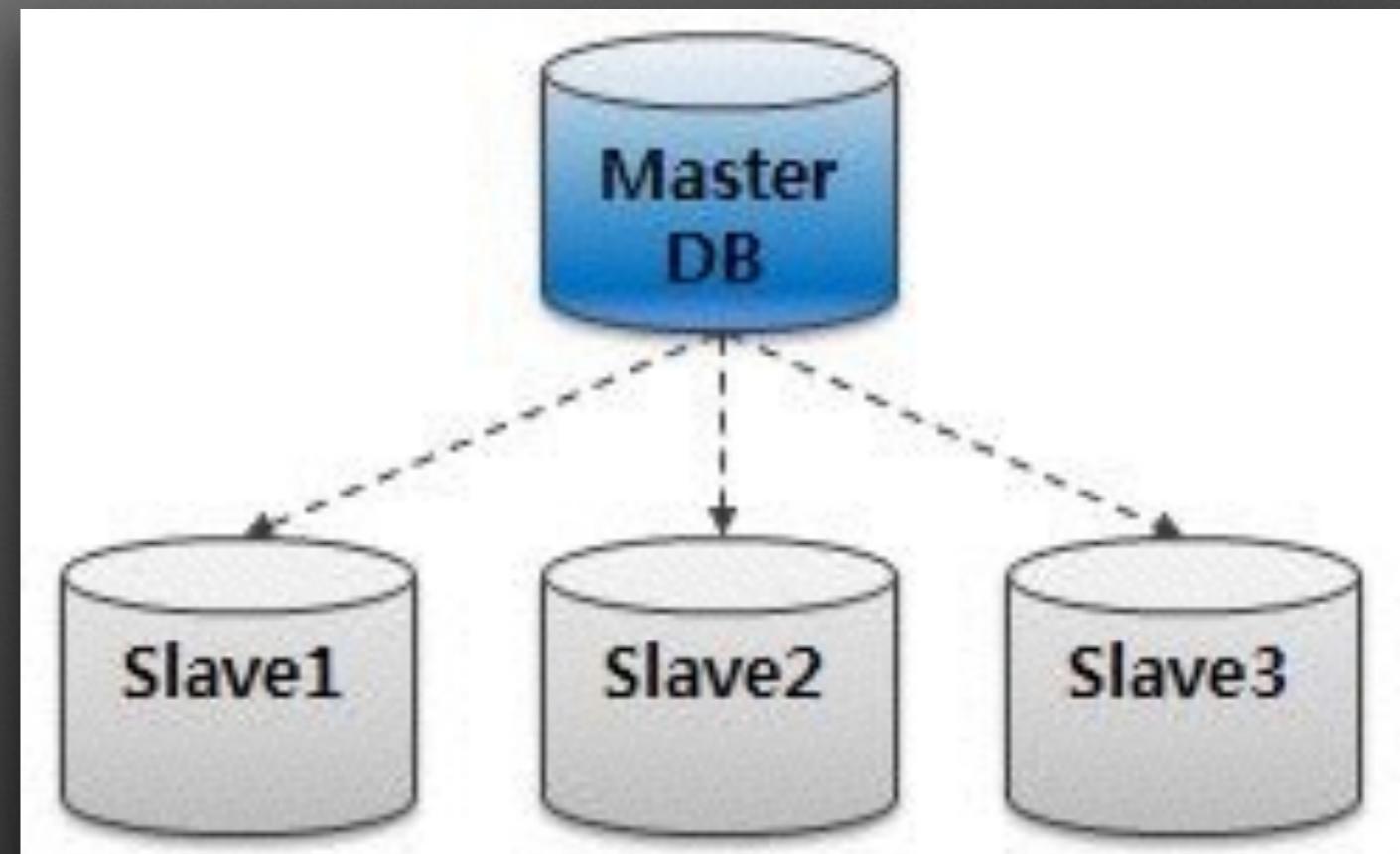
GridFS Benchmark

Conclusion: Reasonable?!

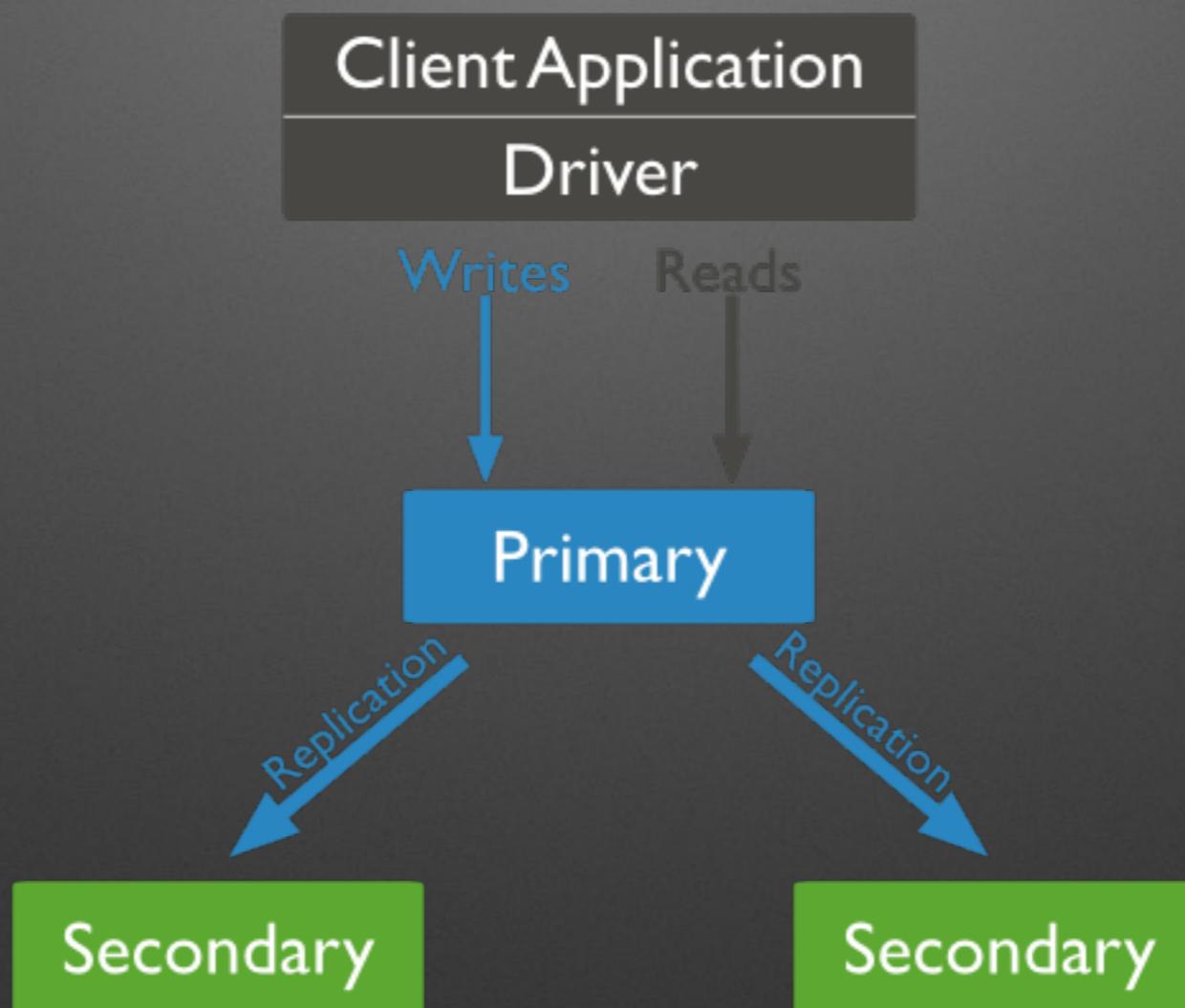
Solution	Requests/second	% Apache FS	% Nginx FS	% Nginx GridFS	% Apache Ruby
Filesystem via Apache	2625.37	-	40.03%	242.22%	4,878.96%
Filesystem via Nginx	6559.31	249.84%	-	605.17%	12,189.76%
GridFS via nginx module	1083.88	41.28%	16.52%	-	2014.27%
Rails metal handler via Passenger	53.81	2.05%	0.82%	4.96%	-

Master-Slave

- 일반적인 Master/Slave 구조
- 읽기 부하 분산
 - Slave는 Read-Only
- Master에 Write 부하 증가
- Master Fail 대응 HA 별도 조치 필요



Replica Sets



Replica Sets

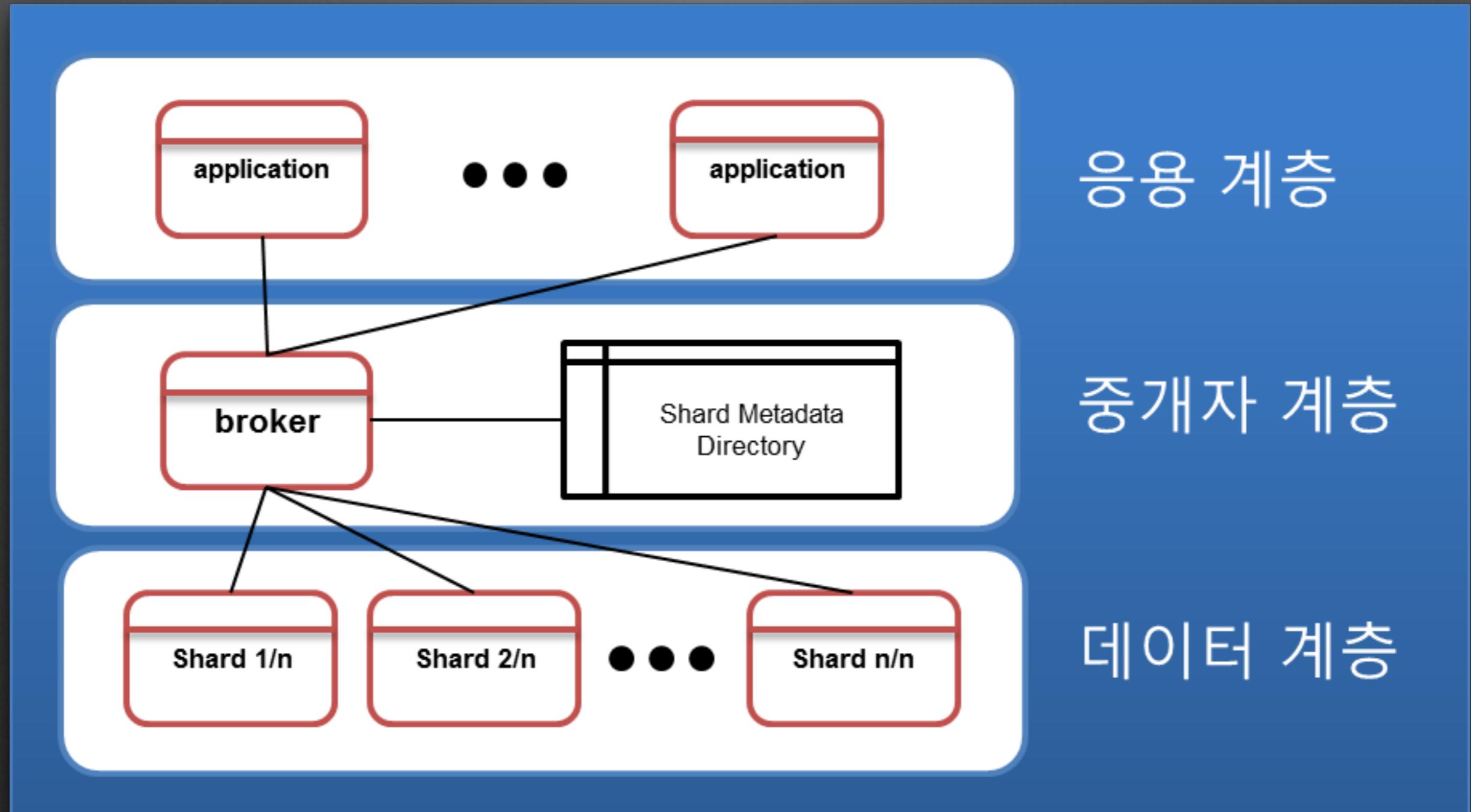
- 동적인 Master Vote 자동 처리
- Master Fail시 Priority에 따른 투표를 통해 Master 대체
- 장애 서버는 복구 후 다시 보조 서버로 편입
- Priority가 0인 후보 서버는 Master 불가 = 백업서버

<https://docs.mongodb.org/v3.0/core/replica-set-members/>

Slave 활용

- 데이터 유실이나 마스터 노드에서 장애 발생시 요청 넘김
- 백업
- 읽기 부하 분산
- 통계 등 데이터 처리용 업무 수행
- Writing에 대한 부하 분산은 불가능

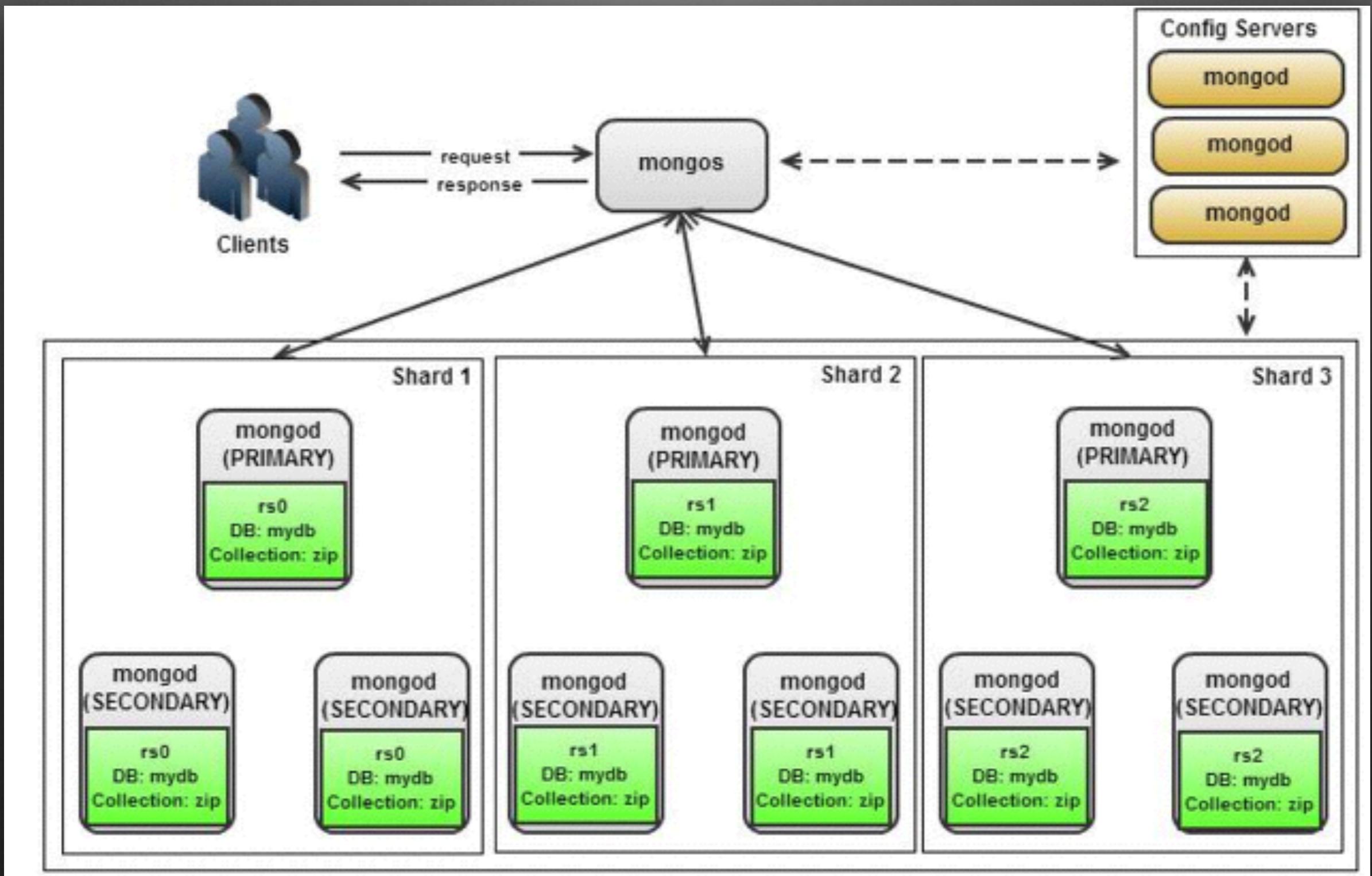
Shard Architect



3-Way Broker

- 형태에 따른 분류
 - 데이터 유형에 따른 테이블 분류
 - 테이블이 분할되지 않으므로 독립성 보장된 쿼리에서 높은 성능
- 키 기반 분류
 - Shard-Key에 따른 분류 (auto/manual)
- Look-Up 테이블 기반 분류
 - Shard-Key의 해쉬값을 통한 랜덤 균등 분할 정책

Shard Architect



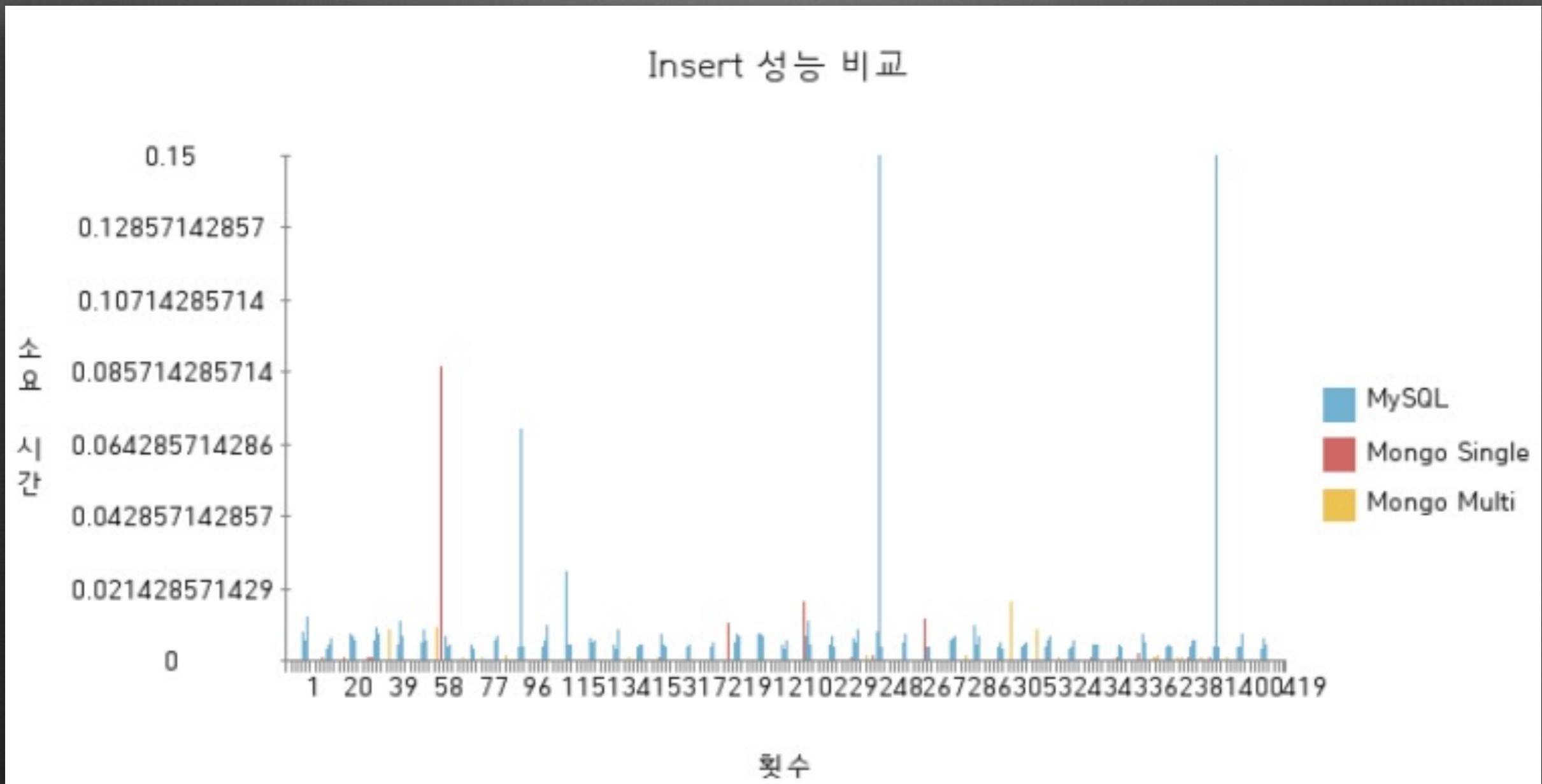
Sharding Condition

- 쓰기(DB Write)가 빈번 할 경우
- 현재 장비에 디스크 공간이 부족할 경우
- 애플리케이션에 영향을 주지 않고 증가하는 부하와 데이터를 처리하기 위해 장비 추가. 즉, Scale-Out

Shard 유의점

- Fragmentation
 - 단편화, 조각화 문제
 - 빈번한 삭제나 업데이트는 단편화를 발생시키고, 실제 데이터 보다 훨씬 많은 메모리를 사용 함
 - 데이터 삭제 또는 이전해도 사용량은 그대로 유지
 - `repairDatabase`를 통해 복구 해야함

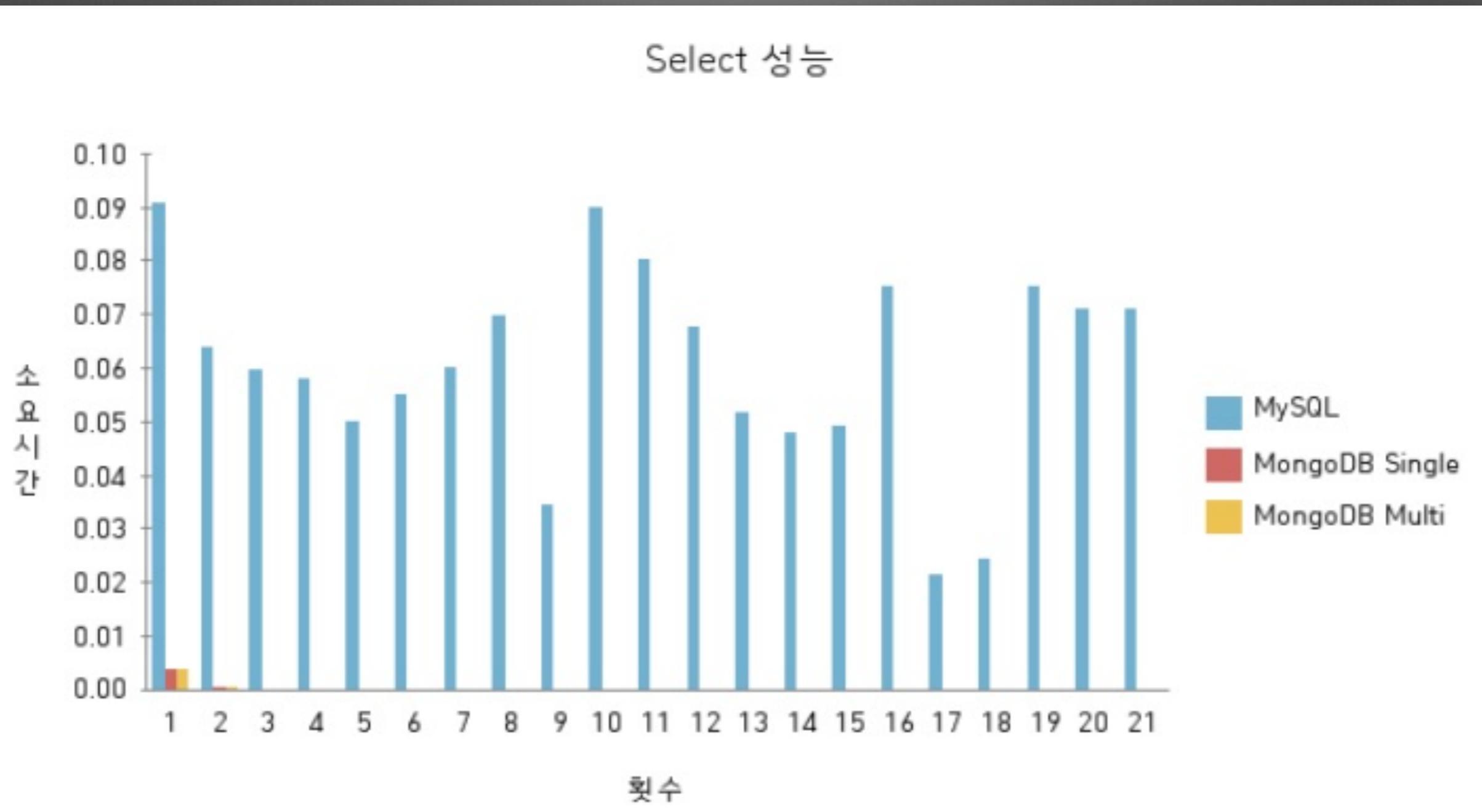
MongoDB vs MySQL



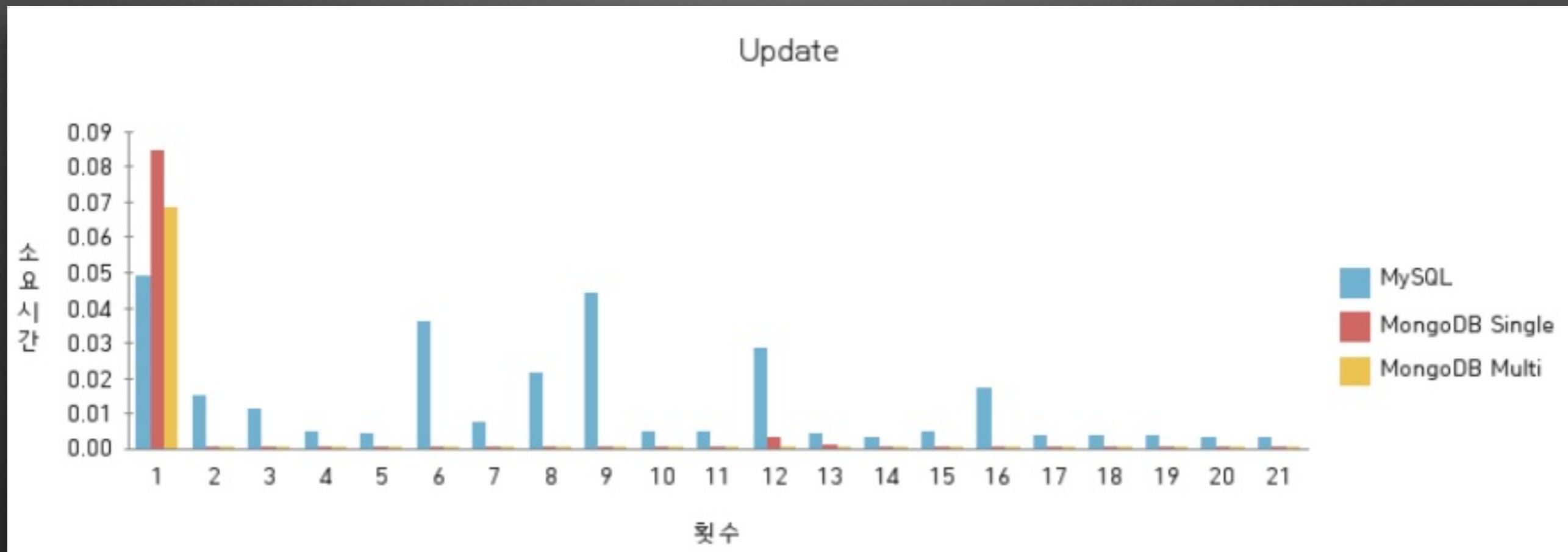
I love MySQL!

MongoDB vs MySQL

Select 성능



MongoDB vs MySQL



MongoDB vs MySQL



MongoDB 활용 유의사항

- 미친듯이 테스트 하라 – Test exhaustively
- RDBMS를 그대로 대체할 수 있다고 가정하지 말라 – Don't assume that what worked for your RDBMS will translate
- 데이터에 대한 일관성과 안전성이 필요한지 고민하라 – Think about the consistency and durability needs of your data

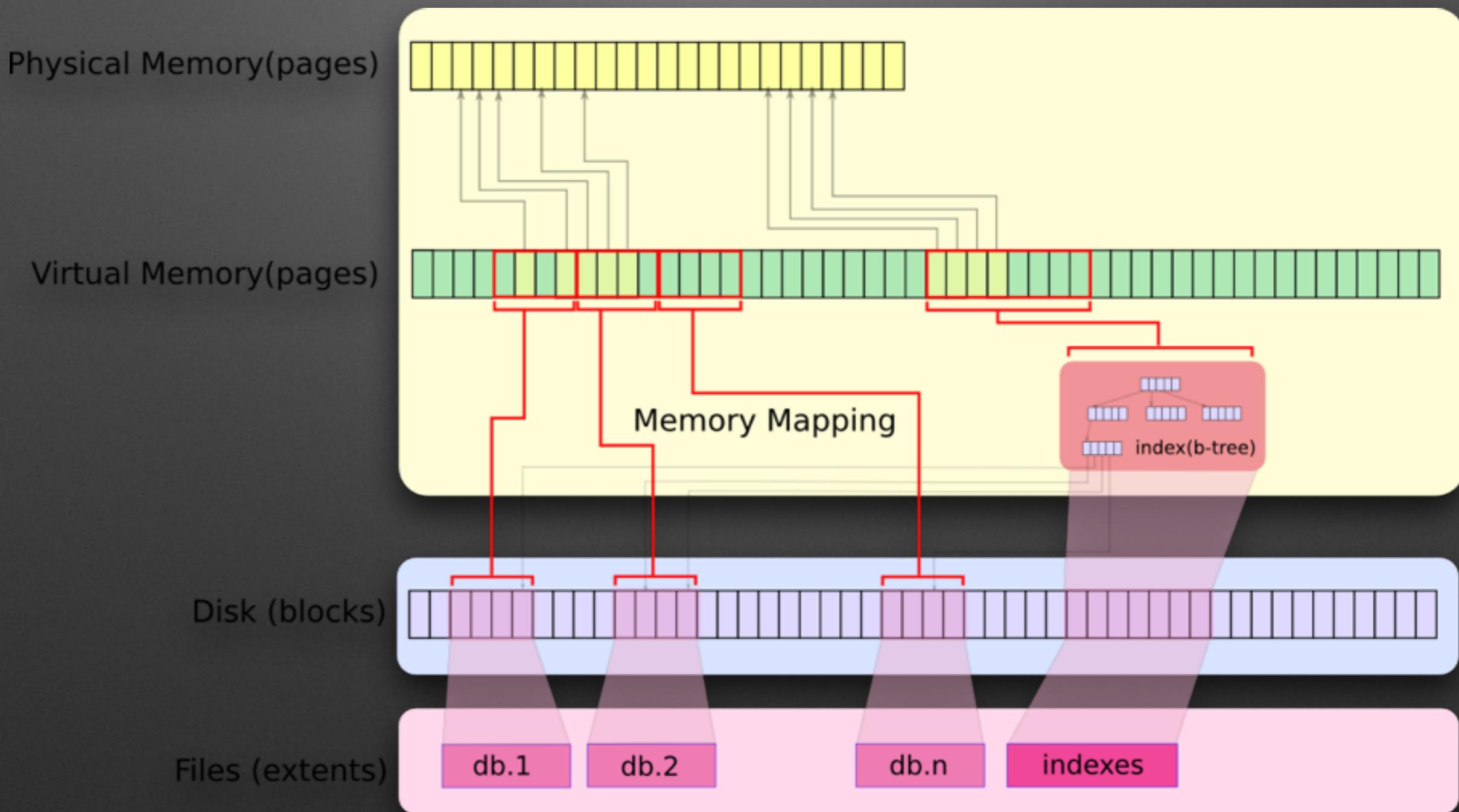
MongoDB 활용 유의사항

- 항상 Replica Sets을 사용하라 – Always use replica sets
- 최신 버전을 유지하라 – Keep current with versions
- MongoDB를 32bit에서 사용하지 말라 – Don't run MongoDB on 32-bit systems
- 기본적으로 Journaling을 사용하라 – Turn journaling on by default

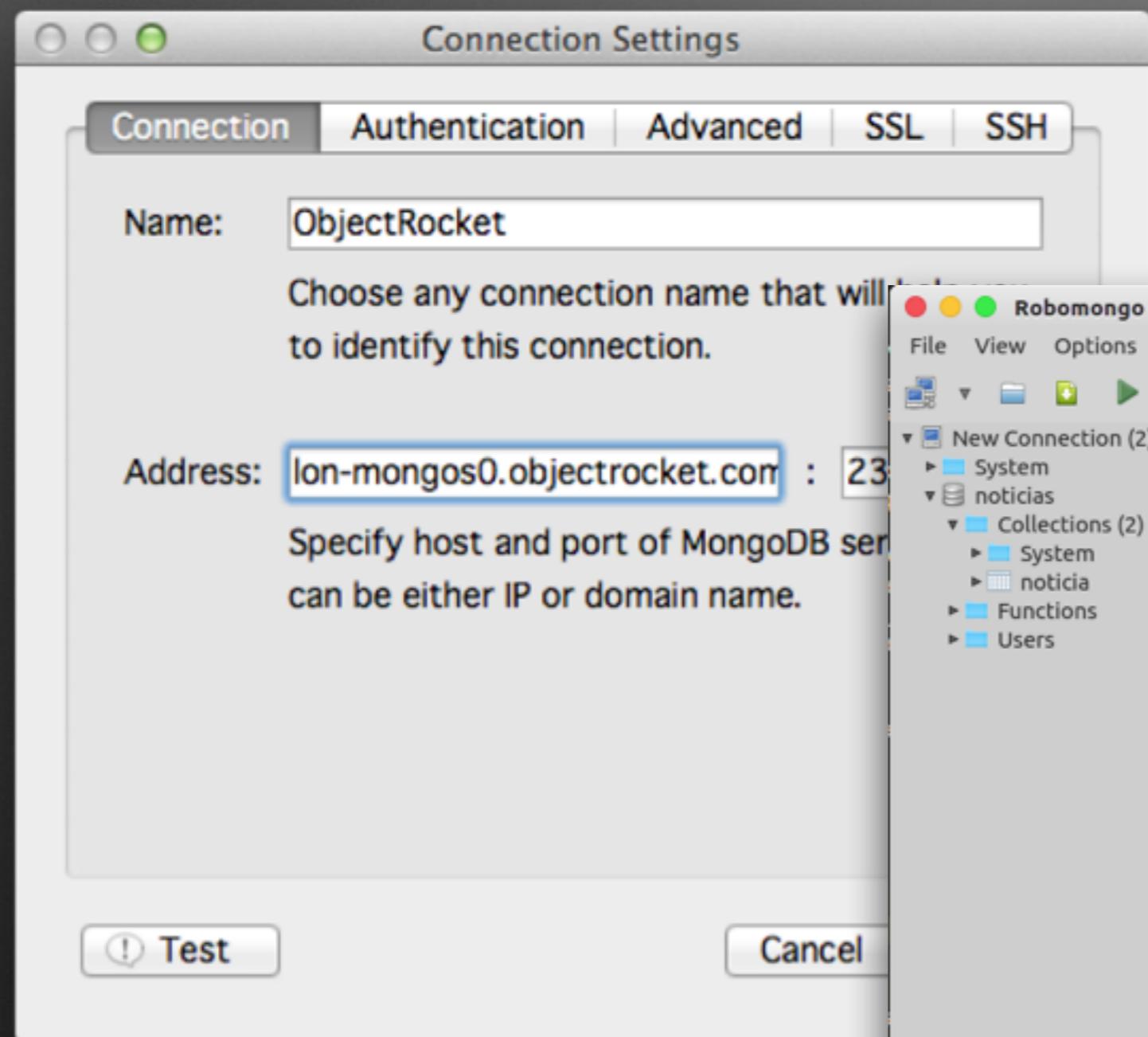
MongoDB 활용 유의사항

- 데이터 파일의 위치를 확인하라 – Mind the location of your data files
- Working Set은 메모리 사이즈에 적합하게 유지하라 – Your working set should fit in memory
- 엄청 많이 사용한다면 Scale Up을 하라 – Scale up if your metrics show heavy use
- 샤딩에 주의하라 – Be careful when sharding

MongoDB 활용 유의사항



RoboMongo



The screenshot shows the RoboMongo interface. The title bar says 'Robomongo 0.8.5'. The left sidebar shows a tree structure with 'New Connection (2)', 'System', 'noticias' (selected), 'Collections (2)' (with 'System', 'noticia', 'Functions', 'Users' children), and 'Functions'. The main area has a query editor window titled 'db.getCollection('noticia')'. It contains the following MongoDB query:

```
1 db.getCollection('noticia').find({})
```

Below the query editor, the results pane shows two documents:

```
1 /* 1 */
2 {
3   "_id" : ObjectId("5621018a30bd142585859a6c"),
4   "titulo" : "hello",
5   "texto" : "world",
6   "imagem" : "aaa"
7 }
8
9 /* 2 */
10 {
11   "_id" : ObjectId("562106b530bd142bba399088"),
12   "titulo" : "teste",
13   "texto" : "teste",
14   "imagem" : "me.gif"
15 }
```

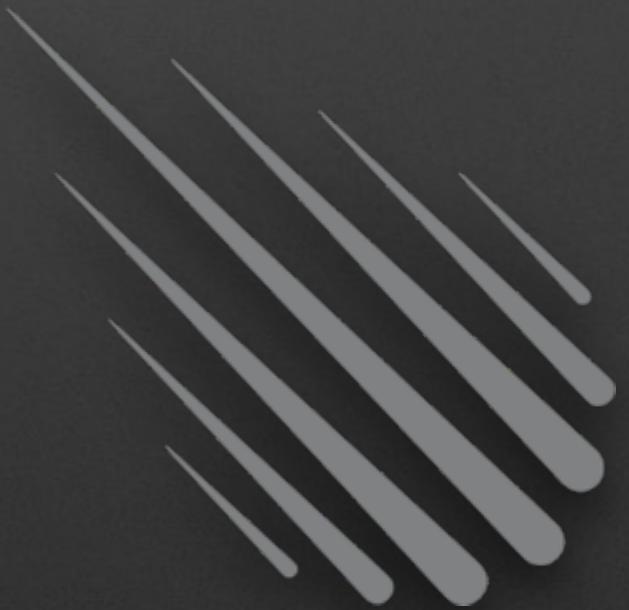
Conclusion

조심히 쓸만하다

안되면 아끼 탓

Meteor.js with MongoDB

A better way to build a web application



A better way to build a web

- 2015.11 기고

2015.12 휴간

Meteo.js 웹애플리케이션을 만드는 더 나은 방법

Meteor.js는 고르밀에서 유추해 볼 수 있듯이 미터어(Meteor)는 자바스크립트로 이뤄진 플랫폼이다. 한 가지 의문을 가질만한 부분은 앞서 나온 기술들을 '웹프레임워크'라고 지칭했던 반면 Meteor.js를 지칭할 때는 '플랫폼'이라고 특정짓고 있다는 사실이다. 이 글은 프레임워크와 플랫폼이라는 용어의 불명확한 만큼이나 설명하기 어려운 주제를 쉽게 전달하기 위해 쓰였다.

Framework	Github Score	Stack Overflow Score	Overall Score
ASP.NET	100	100	100
Node.js	95	95	95
Angular.js	100	10	55
ASP.NET MVC	95	95	91
React	95	95	95
Meteor	95	17	55

(그림 1) 웹프레임워크 사용성 순위. Mon, 2015년 8월, 출처: hotuniversity.com/

유수의 프레임워크들과 함께 6위에 Meteor.js가 올랐다. (그림 1)은 깃허브(Github)와 스택오버플로우(Stack Overflow) 이용율을 기반으로 측정한 데이터다. 학문적인 연구나 이미 운영중인 레거

시 시스템에 대한 기준이 아닌 세계에서 활발히 활동중인 현지 개발자들의 관심 프레임워크 순위라고 명명하는 것이 어울린 듯 싶다. 아직 생소하지만 Meteor.js가 어느덧 상위에 기록될 만큼 영향을 갖겠다고 이야기 할 수 있다. 어떠한 이유로 저렇게 사랑 받고 있는지 한 번 알아보자.

Meteor.js 특징

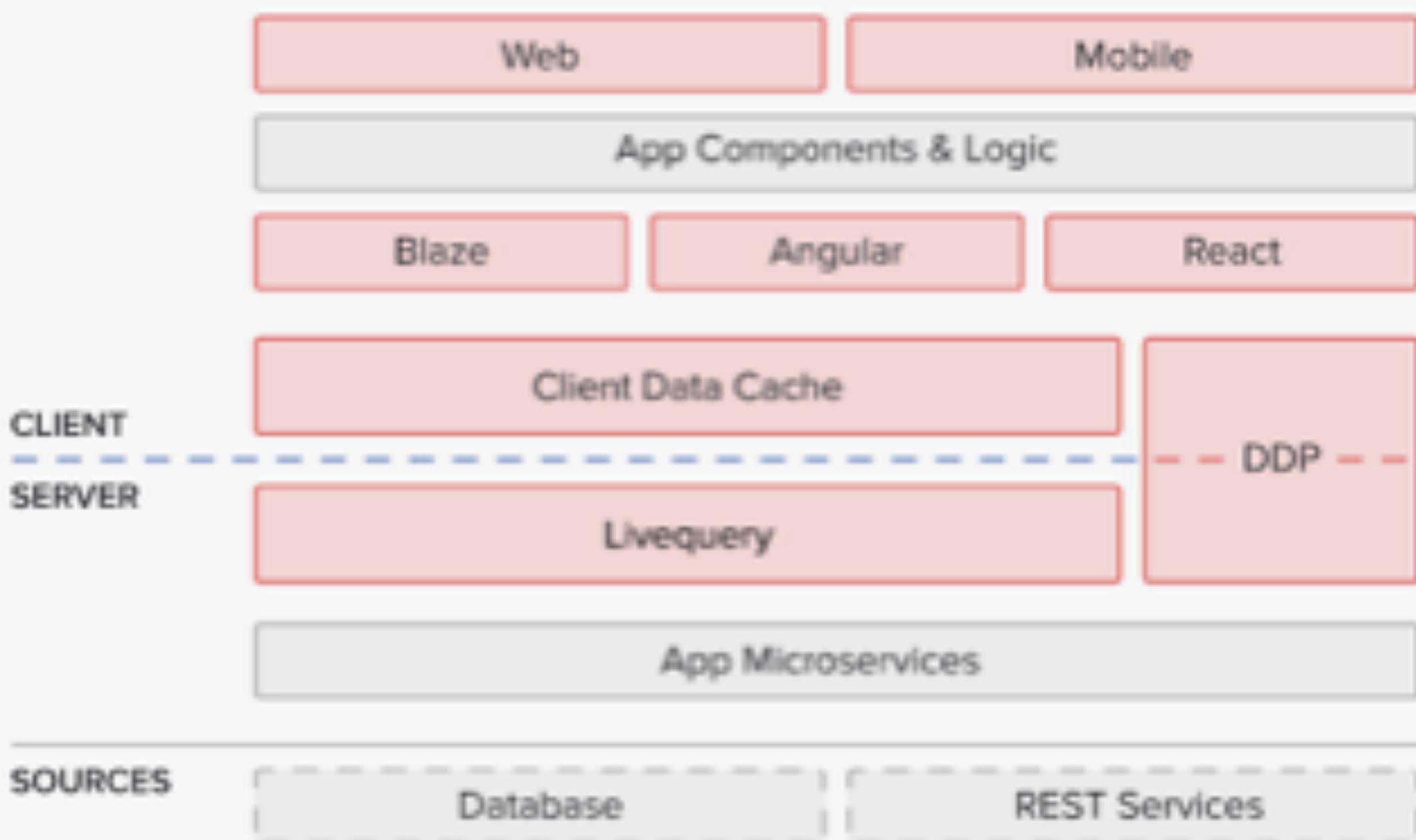
- 풀 스택 프레임워크(Full Stack Framework, Platform)
Angular.js, React.js로 대표되는 트렌디한 자바스크립트 셀프레

Meteor.js

Rankings

Framework	Github Score	Stack Overflow Score	Overall Score
ASP.NET		100	100
Ruby on Rails	95	98	96
AngularJS	100	93	96
ASP.NET MVC		93	93
Django	90	92	91
Meteor	96	77	86

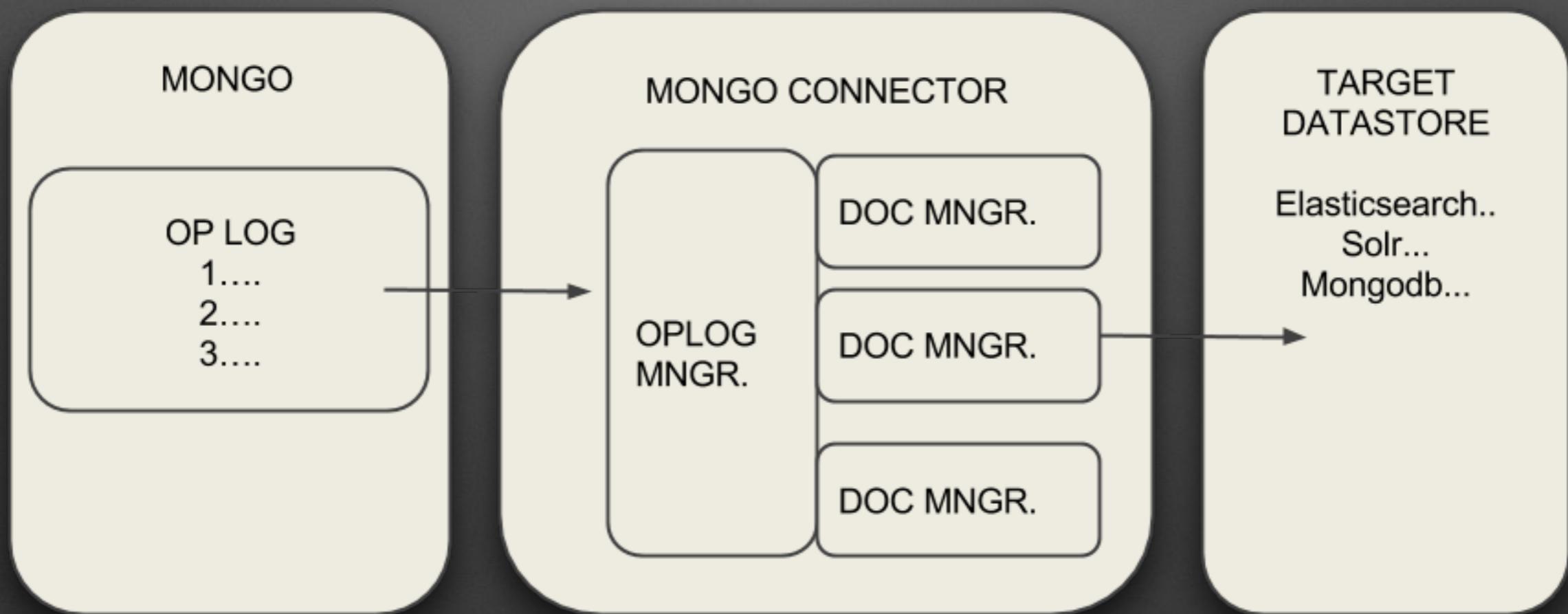
Meteor.js



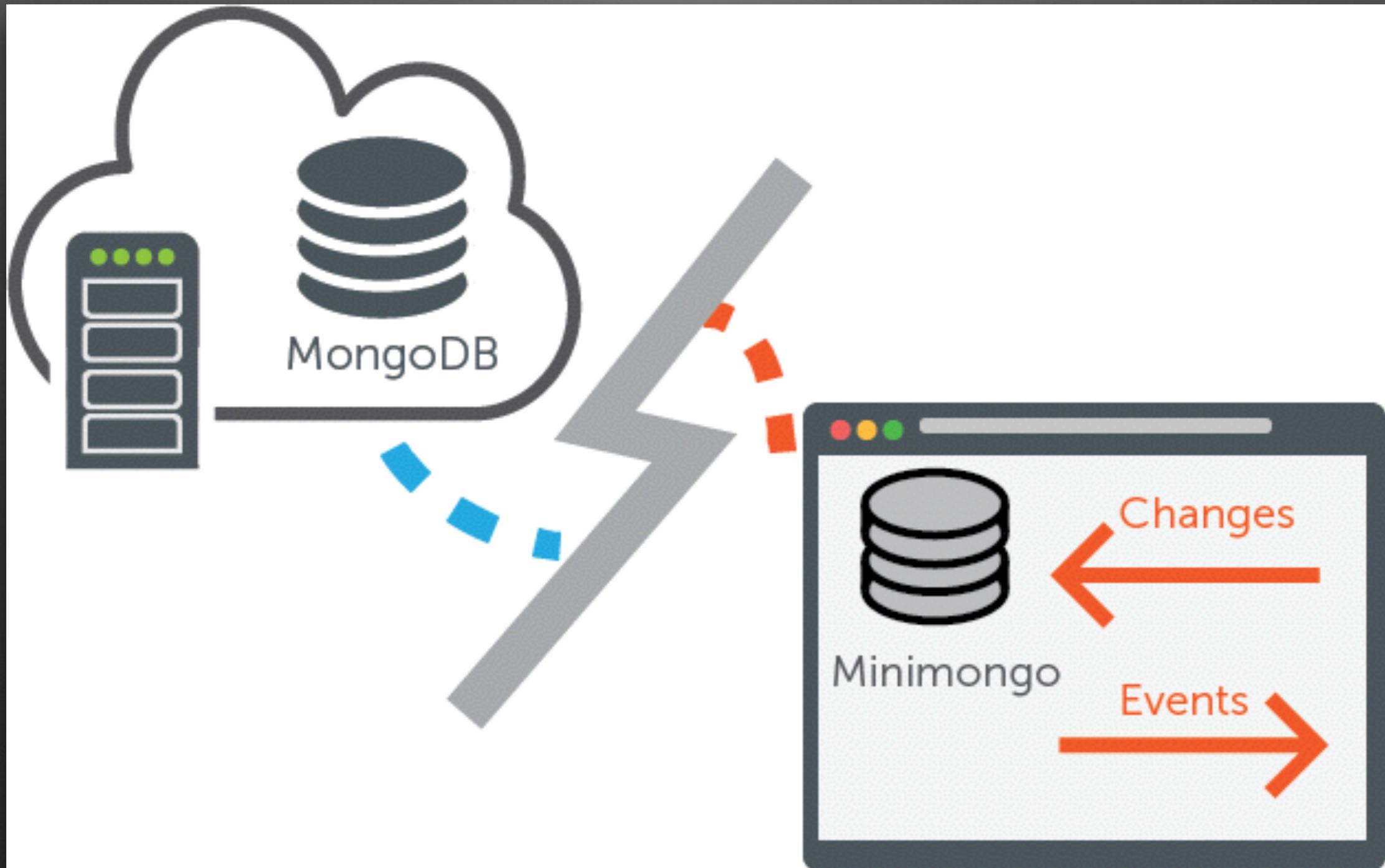
Benefits

- Full Stack Framework (Platform)
- Full javascript
- Easy Install / Usage / Package / Deployment
- Auto Install with MongoDB
 - Reactive
 - Latency Comprehension

Reactive with OPLOG



Latency Comprehension



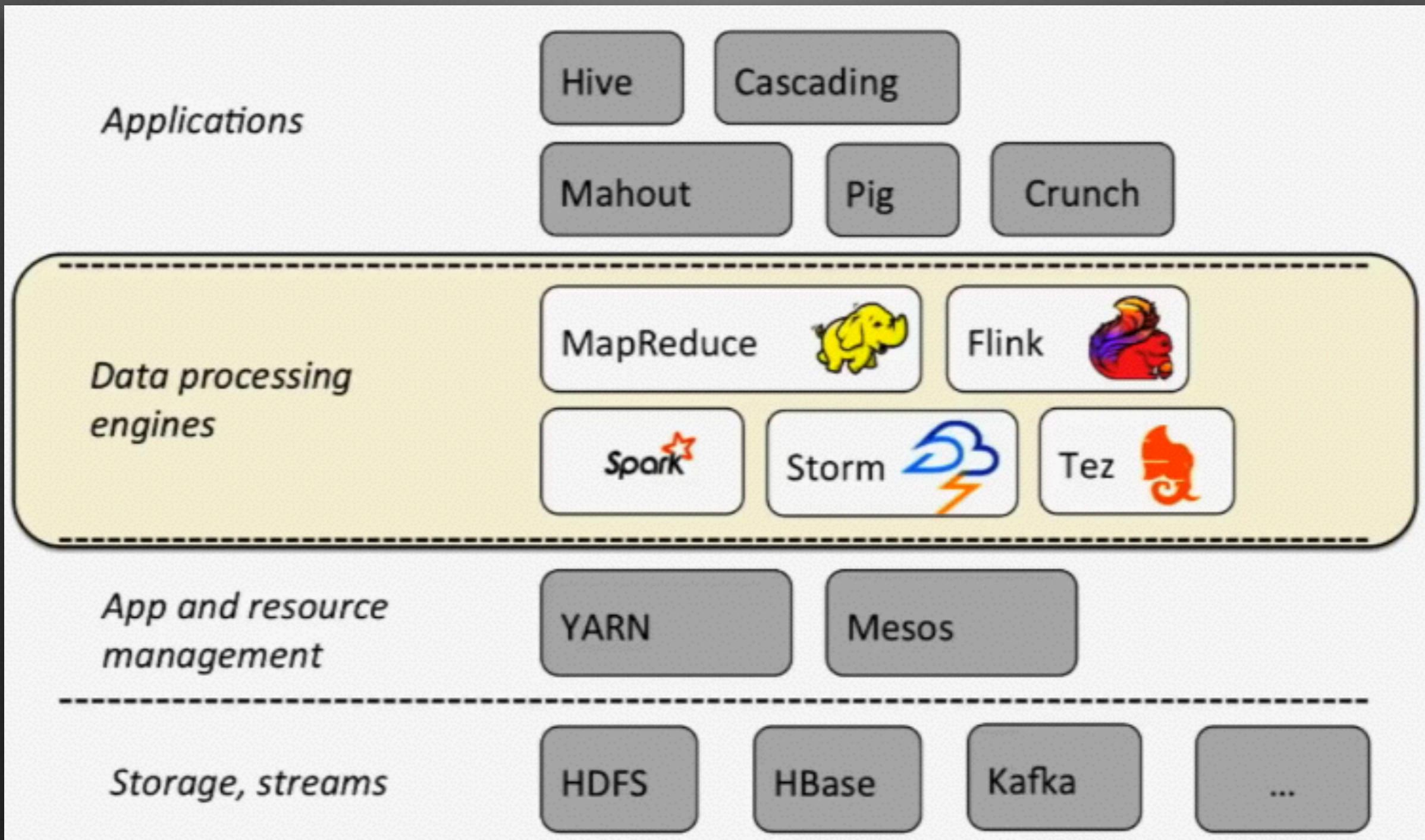
Spark

Overview

- 범용적 목적의 분산 고성능 클러스터링 플랫폼
(General purpose high performance distributed platform)
- Map & Reduce (cf. Hadoop)
- Streaming 데이터 핸들링 (cf. Apache Storm)
- SQL 기반의 데이터 쿼리 (cf. Hadoop의 Hive)
- 머신 러닝 라이브러리 (cf. Apache Mahout)

어차피 여기저기서 맹 듣는 소리들

Spark in Hadoop



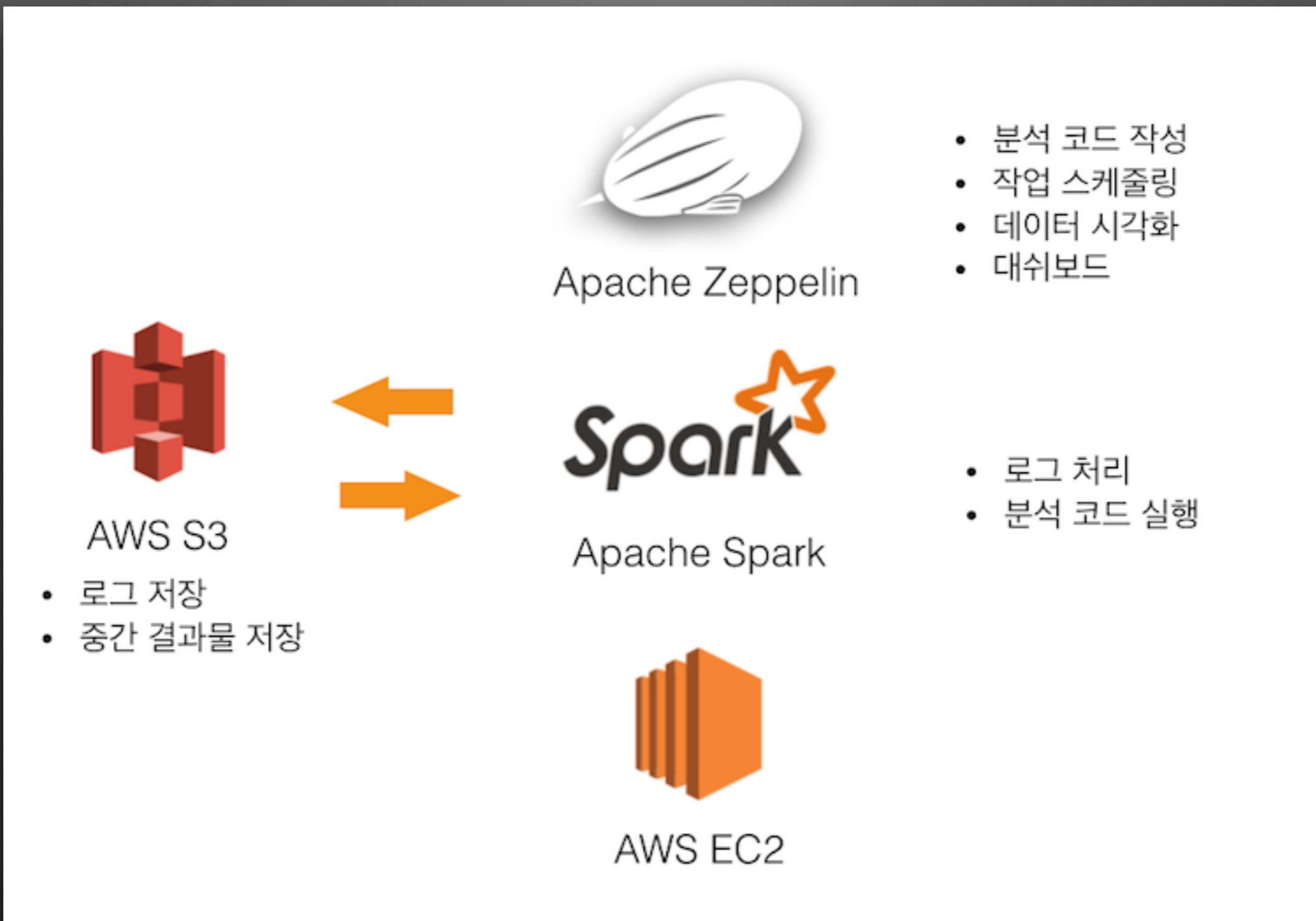
General Hadoop Process

- ETL로 데이터 로딩
- MR로 데이터 정체
- RDBMS, NoSQL로 구성된 OLAP등의 시스템에 임시 저장
- Storm 등으로 실시간 분석
- 리포팅 툴로 그래프 표현

VCNC Hadoop Arch.



VCNC Spark Arch.



Hadoop VS Spark

	기존(Hadoop)	현재(Spark)
일일 배치 분석	코드 작성 및 관리가 어려움	Zeppelin의 Schedule 기능을 통해 수행 Interactive shell로 쉽게 데이터를 탐험 오류가 생긴 경우에 shell을 통해 손쉽게 원인 발견 및 수정 가능
Ad-hoc(즉석) 분석	복잡하고 많은 코드를 짜야 함 분석 작업에 수 일 소요	Interactive shell 환경에서 즉시 분석 수행 가능
Dashboard	별도의 사이트를 제작하여 운영 관리가 어렵고 오류 대응 힘듦	Zeppelin report mode 사용해서 제작 코드가 바로 시각화되므로 제작 및 관리 수월
성능	일일 배치 분석에 약 8시간 소요	메모리를 활용하여 동일 작업에 약 1시간 소요

Spark Pros.

- Include (almost) all of Hadoop Stack
- Speed
- Language: Scala, JAVA, Python, R ...
- DB: HDFS, S3, Cassandra, MongoDB ...

SQL on Hadoop vs Spark

- SQL on Hadoop(Hive, 타조, 임팔라)은 SQL에 친숙한 사용자를 타겟팅
- Spark는 복잡한 Map-Reduce에 지친 사용자들이 주요 타겟팅

Language Diff

- Closure in Python

```
lines = sc.textFile(...)
```

```
lines.filter(lambda s: "ERROR" in s).count()
```

- Closure in Scala

```
val lines = sc.textFile(...)
```

```
lines.filter(s => s.contains("ERROR")).count()
```

- Closure in JAVA

```
JavaRDD lines = sc.textFile(...);
```

```
lines.filter(new Function<String, Boolean>() {
```

```
    Boolean call(String s) {
```

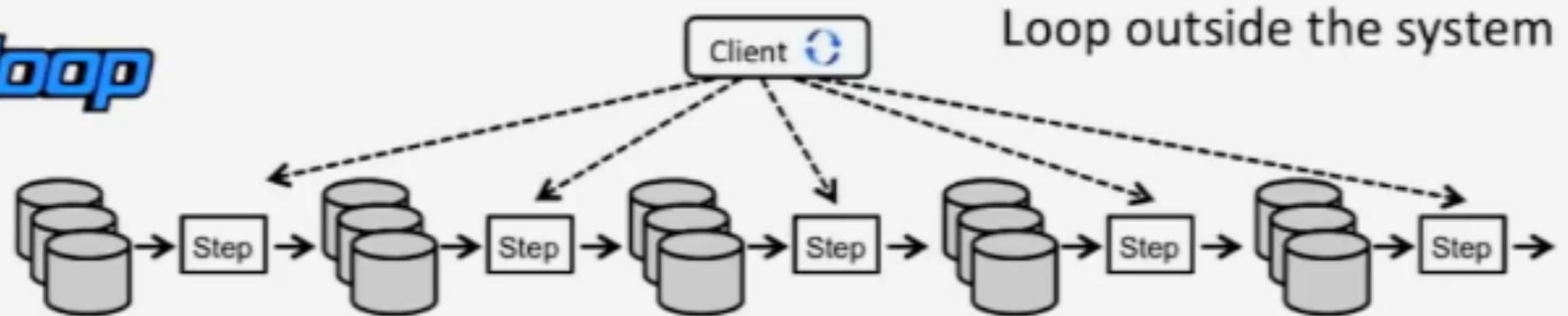
```
        return s.contains("error");
```

```
    }
```

```
}).count();
```

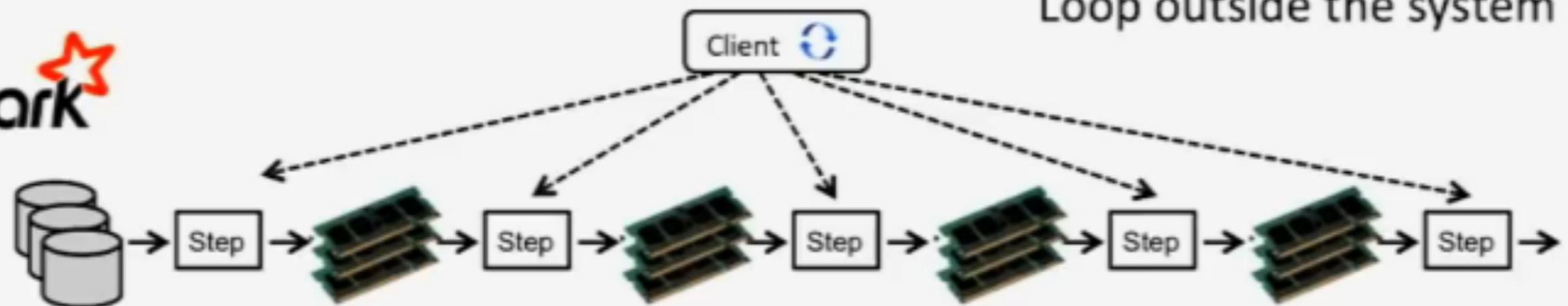
I love Java!

Iteration of Spark & Hadoop



→ Move data through disk and network (HDFS)

I love RAM!



→ User can cache data in memory

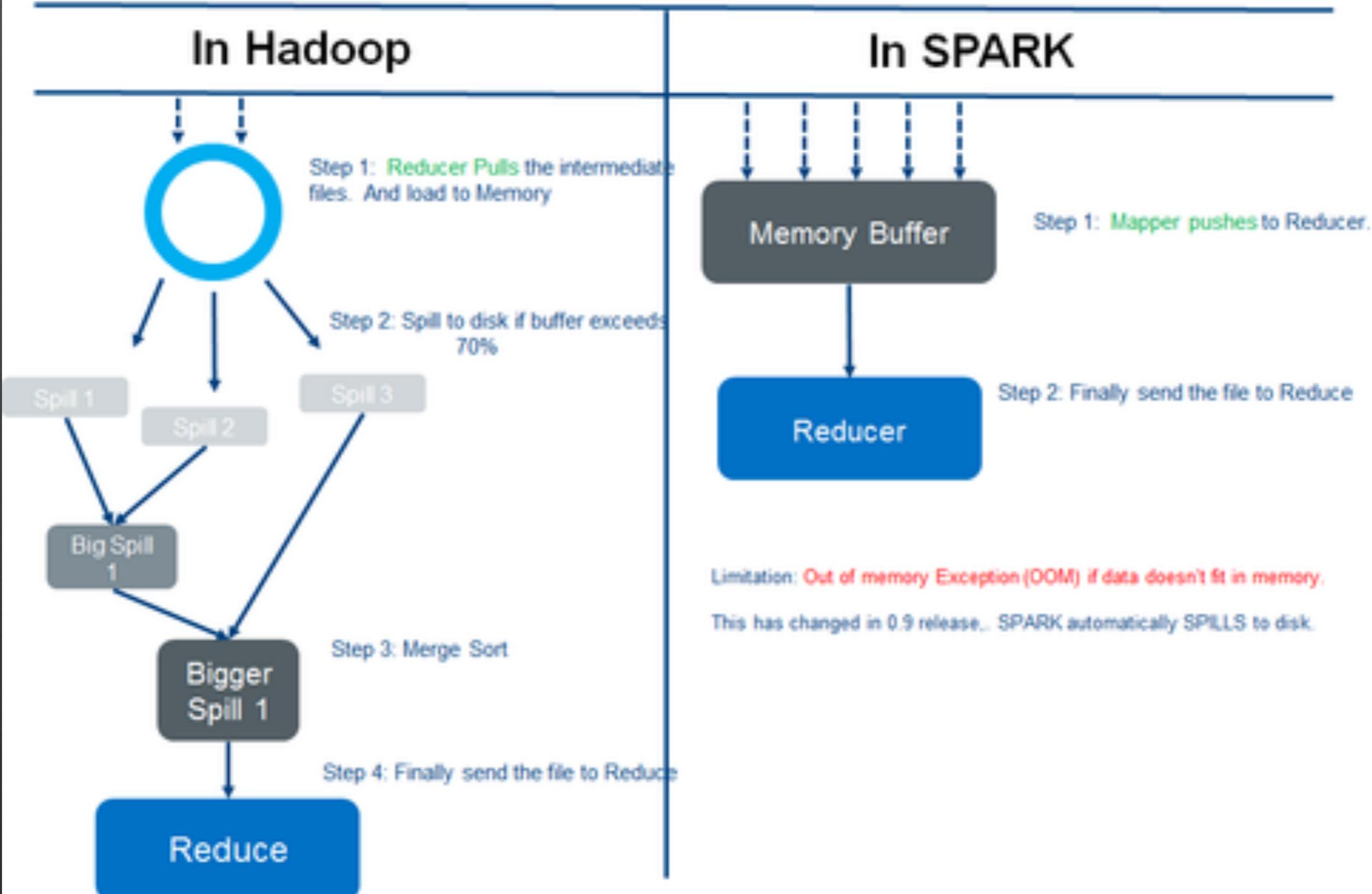
Map, vs Hadoop

Map side - Shuffle Phase Differences...

In Hadoop	In SPARK: Initial design
<p>The diagram shows a 'Mapper' box at the top, connected by an arrow to a circular buffer icon. Three arrows point from the buffer to three boxes labeled 'R1', 'R2', and 'R3'. Below the buffer are five boxes labeled 'Spill 1', 'Spill 2', 'Spill 3', 'Spill 4', and 'Spill 5'. An arrow points from 'Spill 2' down to a stack of three boxes labeled 'R1', 'R2', and 'R3'. To the right of the diagram, steps are listed:</p> <ul style="list-style-type: none">Step 1: Emit the Map OutputStep 2: Store data in circular BufferStep 3: Spill the buffer data to disk, if it's 80% complete.Step 4: Sort and partition the data. Create one big file per node. Partitioned based on no of reducers i.e., R1,R2,R3	<p>The diagram shows two 'Mapper1' and 'Mapper2' boxes at the top, each with three arrows pointing to three boxes labeled 'R1', 'R2', and 'R3'. To the right of the diagram, steps are listed:</p> <ul style="list-style-type: none">Step 1: Emit the Map OutputStep 2: Create R(Reducer) shuffle files per Mapper. Data usually stay in OS Buffer Cache. Some written to disk. Hence the Write/Read is at memory speed.Eg: if M = 2000 and R = 6000 i.e $2000 \times 6000 = 12$ million Around 12 Million Shuffle files !! Performance Problems.
	<p>The diagram shows five 'Mapper' boxes (1-5) at the top, each with an arrow pointing to a stack of four boxes labeled 'C1', 'C2', 'C3', and 'C4'. Arrows from 'Mapper 1' and 'Mapper 2' point to 'C1' and 'C2' respectively. Arrows from 'Mapper 3' and 'Mapper 4' point to 'C3'. An arrow from 'Mapper 5' points to 'C4'. Below the cores are three boxes labeled 'R1', 'R2', and 'R3'. Arrows from 'C1' and 'C2' point to 'R1' and 'R2' respectively. An arrow from 'C3' points to 'R3'. To the right of the diagram, text provides context:</p> <p>R=Mapper C=Core Eg: if R = 2000 and C = 4 $2000 \times 4 = 8000$ Around 8000 Shuffle files. Huge improvement in performance. SET: <code>spark.shuffle.consolidatefiles</code> To "true"</p> <p>Node1 with 4 Cores</p>

Reduce, vs Hadoop

Reduce side - Shuffle Phase Differences...



Fault-Tolerant

- Data Replication: 어라?, 더 느리다.
- Check-Point 설정: 어라?! 더욱 느리다.
- Insert Only, No Modify
 - => RDD(Resilient Distributed DataSet) 모델

RDD

- Immutable
- Distributed
- Lazily evaluated
- Type inferred
- Cacheable

General Data -> RDD 일방향 변환

RDD Operator

- Transformations
 - Map, Reduce, Join 등 풍부한 명령어로 Data를 보다 자유롭게 컨트롤
 - 수행 플랜(Lineage)만 계획, 실제 연산은 이루어지지 않음
- Actions: Output을 산출하기 위해 실제 연산
- Lazy-Execution에 따른 이득
 - 분산 자원 배치 상황을 고려, 최적의 코스로 실행

Node Distribution

- Wide-Dependency
 - 노드 분산에 의한 네트워크 비용 증대
 - Crash/FailOver 발생 시 복구 비용 증대
- Narrow-Dependency
 - 노드 분산 방지로 네트워크 비용 감소
 - Crash/FailOver 발생 시 복구 비용 감소

물론 이딴걸 몰라서 못하는건 아니다..

Job Scheduling

- Directed Acyclic Graph에 의거 실시
- 중복/반복 파티션은 연산 Skip
- 수행 노드는 Locality 고려 선정

Job Scheduling

Spark Jobs (?)

Total Uptime: 2.2 min

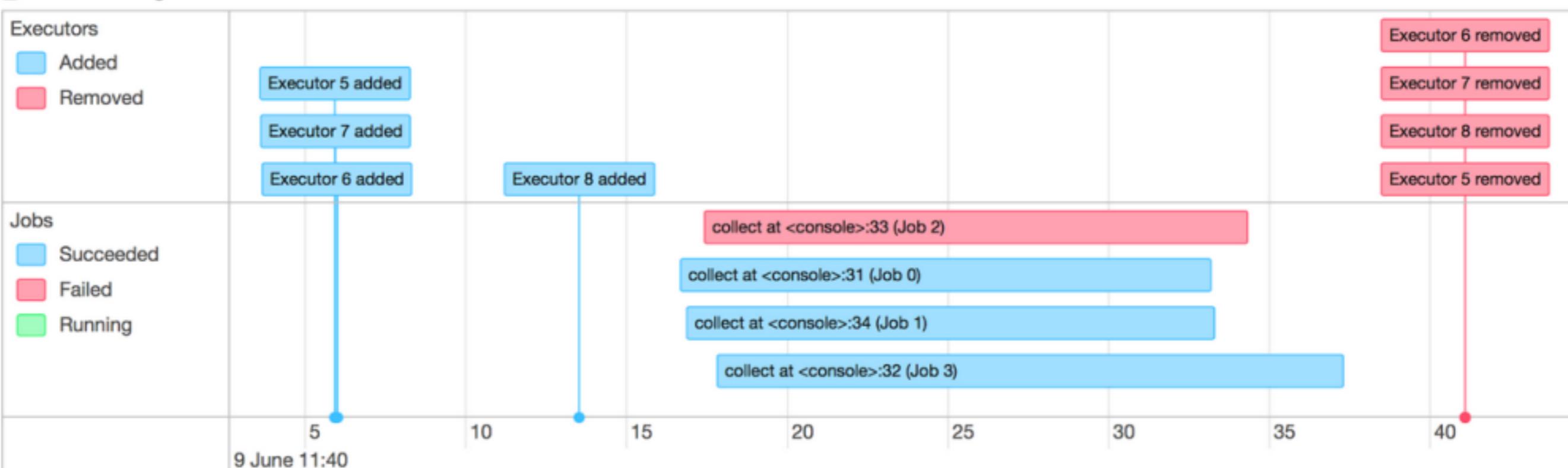
Scheduling Mode: FIFO

Completed Jobs: 3

Failed Jobs: 1

▼ Event Timeline

Enable zooming



Job Scheduling

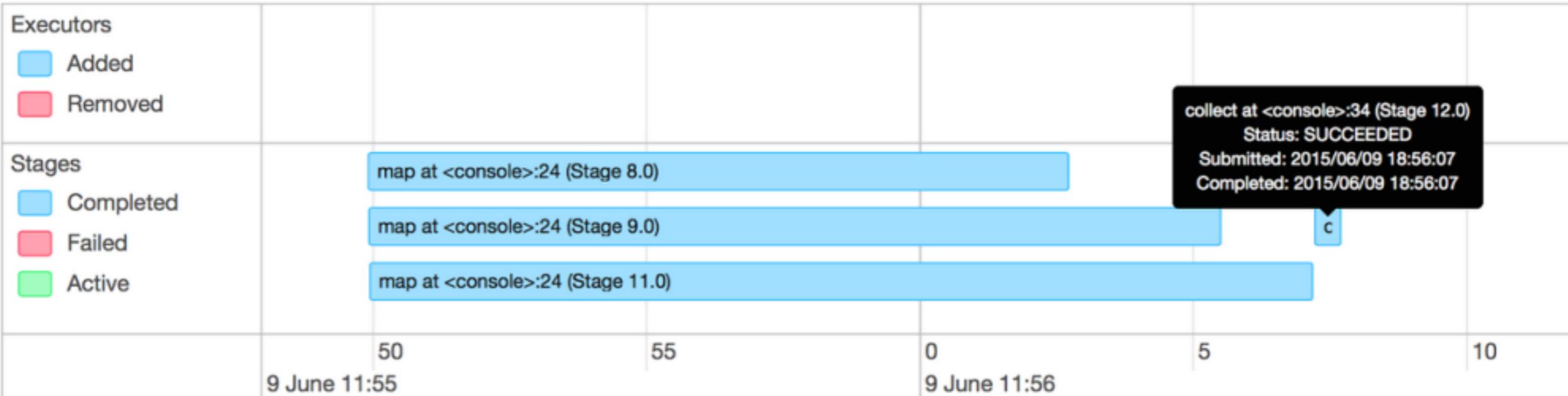
Details for Job 1

Status: SUCCEEDED

Completed Stages: 5

▼ Event Timeline

Enable zooming



Details for Stage 11 (Attempt 0)

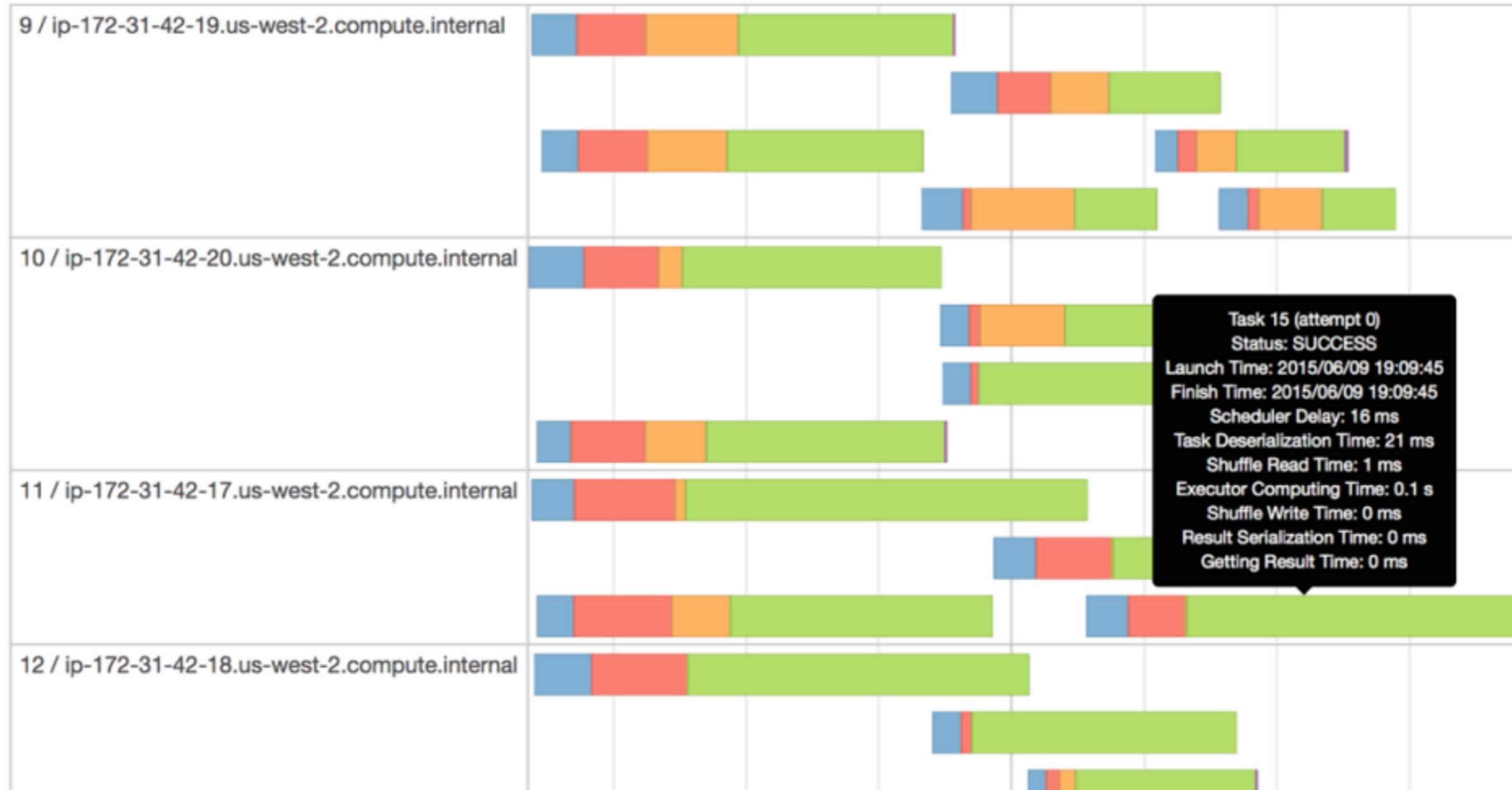
Total Time Across All Tasks: 2 s

Shuffle Read: 200.2 KB / 13839

- ▶ DAG Visualization
- ▶ Show Additional Metrics
- ▼ Event Timeline
- Enable zooming

█ Scheduler Delay
█ Task Deserialization Time
█ Shuffle Read Time

█ Executor Computing Time
█ Shuffle Write Time
█ Result Serialization Time



Job Scheduling

Spark Jobs (?)

Total Uptime: 1.2 min

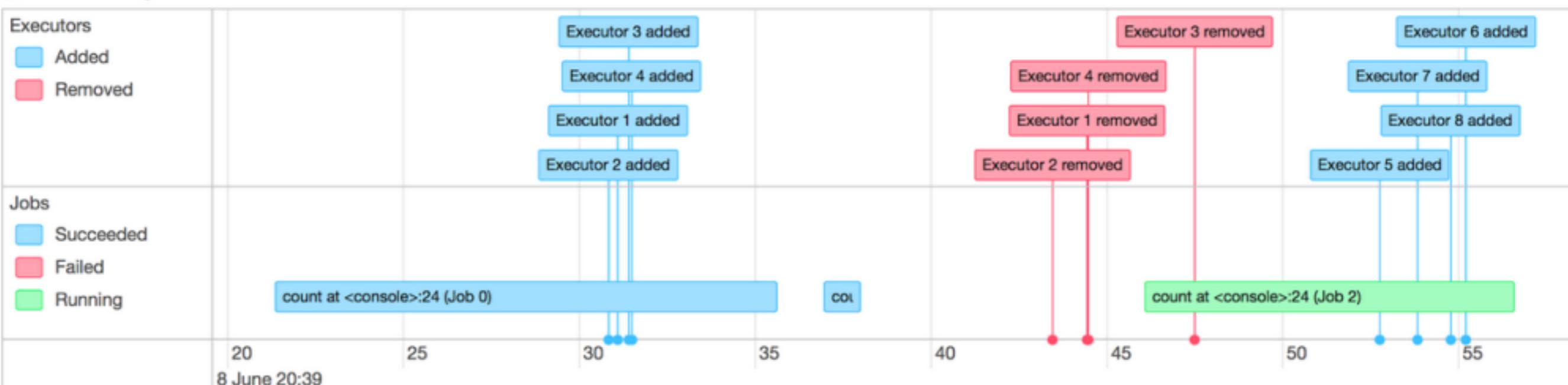
Scheduling Mode: FIFO

Active Jobs: 1

Completed Jobs: 2

▼ Event Timeline

Enable zooming



DAG

Direct Acyclic Graph

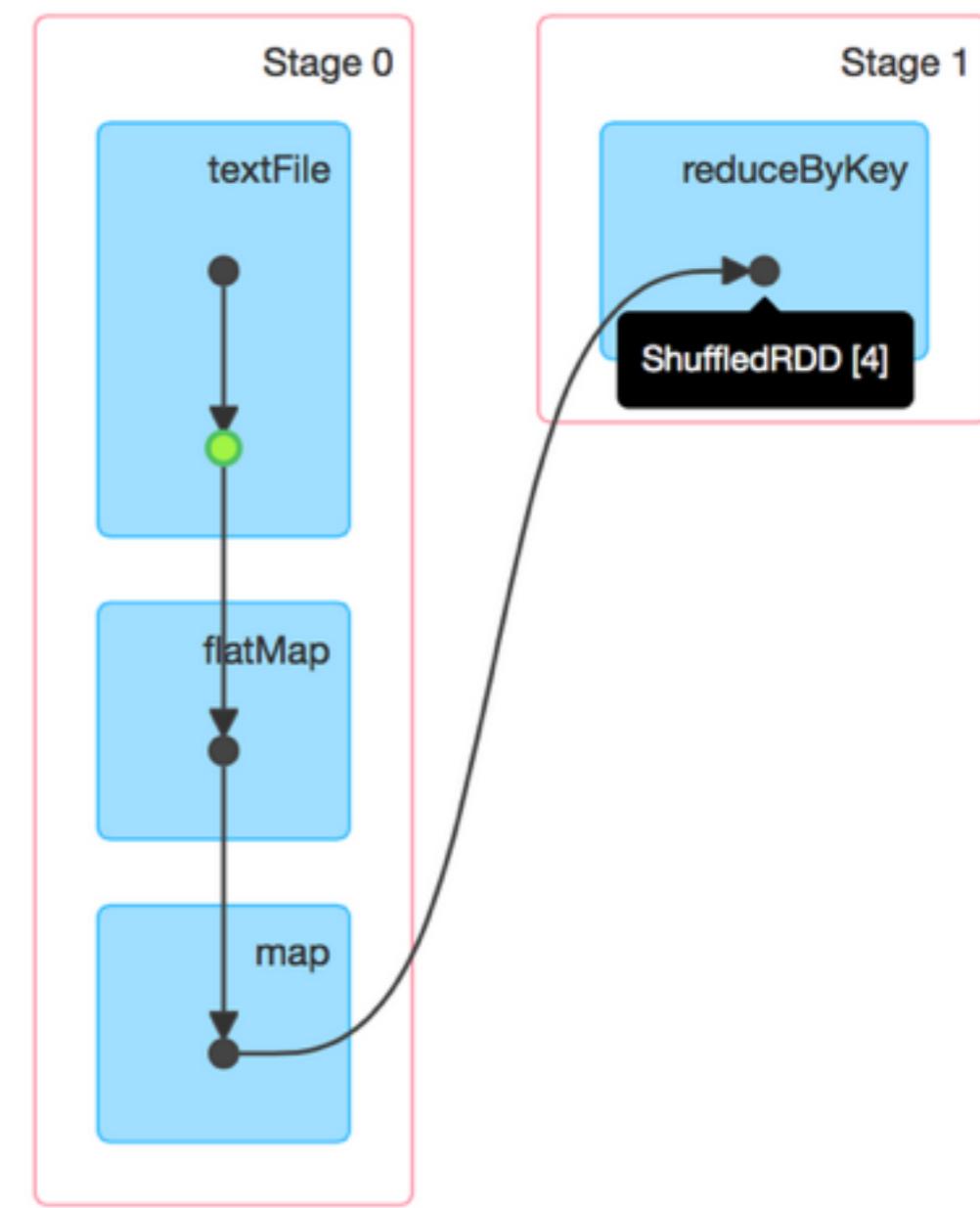
Sequence on a simple job

Details for Job 0

Status: SUCCEEDED

Completed Stages: 2

- ▶ Event Timeline
- ▼ DAG Visualization



Exception Handling

- You need more Pylo.. Memories.
 - LRU(Least Recently Used) 순으로 파티션 관리
- Fault Recovery / FailOver
 - 기록된 수행 로그(Lineage)에 따라 Recovery
 - 파티션 자체 파괴시 Secondary Node에서 복구

Hadoop Word Counting

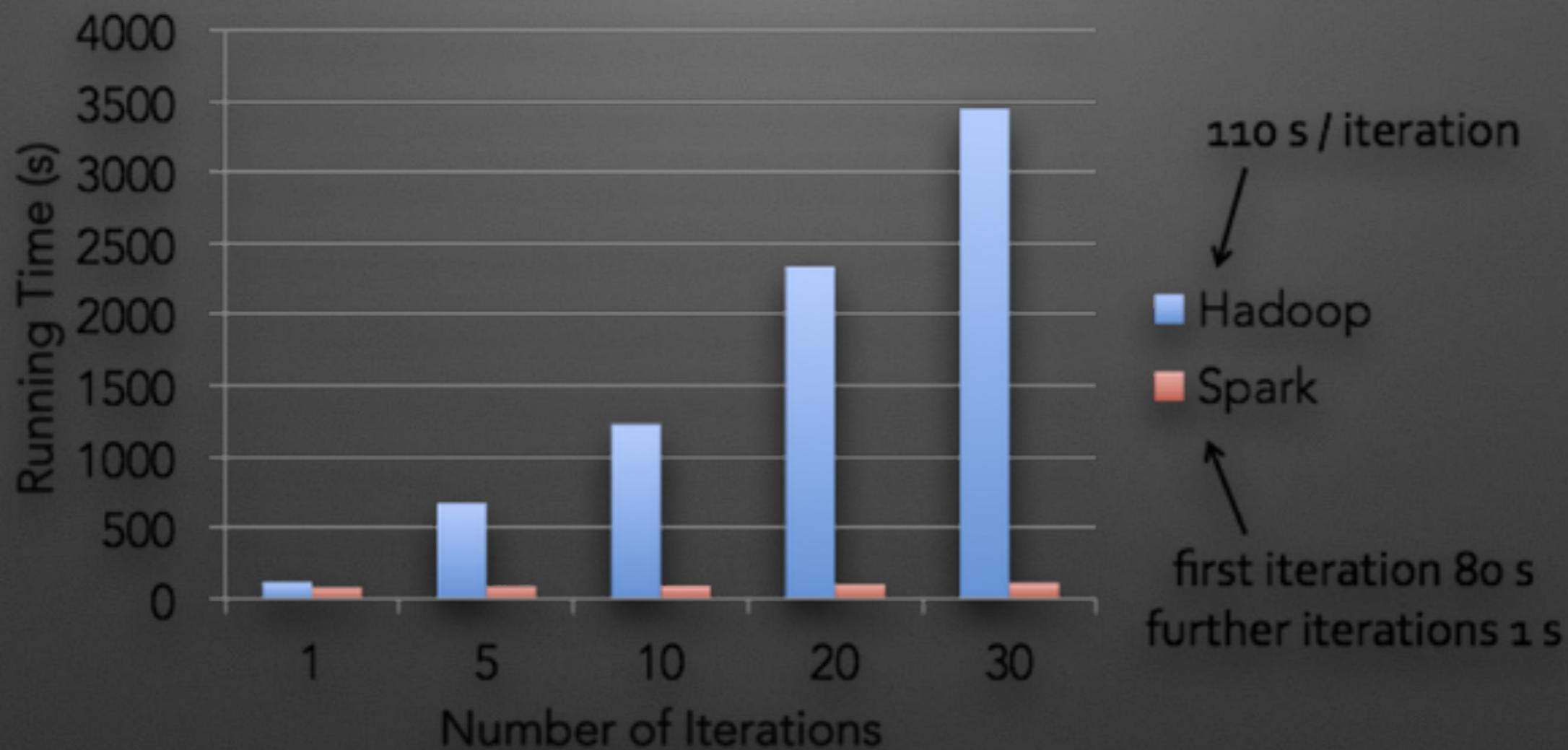
```
public static class WordCountMapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value,
                    OutputCollector<Text, IntWritable> output,
                    Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer itr = new StringTokenizer(line);
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            output.collect(word, one);
        }
    }
}
public static class WordCountReduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator values,
                      OutputCollector<Text, IntWritable> output,
                      Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

Spark Word Counting

```
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: wordcount <file>", file=sys.stderr)
        exit(-1)
    sc = SparkContext(appName="PythonWordCount")
    lines = sc.textFile(sys.argv[1], 1)
    counts = lines.flatMap(lambda x: x.split(' '))
                  .map(lambda x: (x, 1))
                  .reduceByKey(add)
    output = counts.collect()
    for (word, count) in output:
        print("%s: %i" % (word, count))

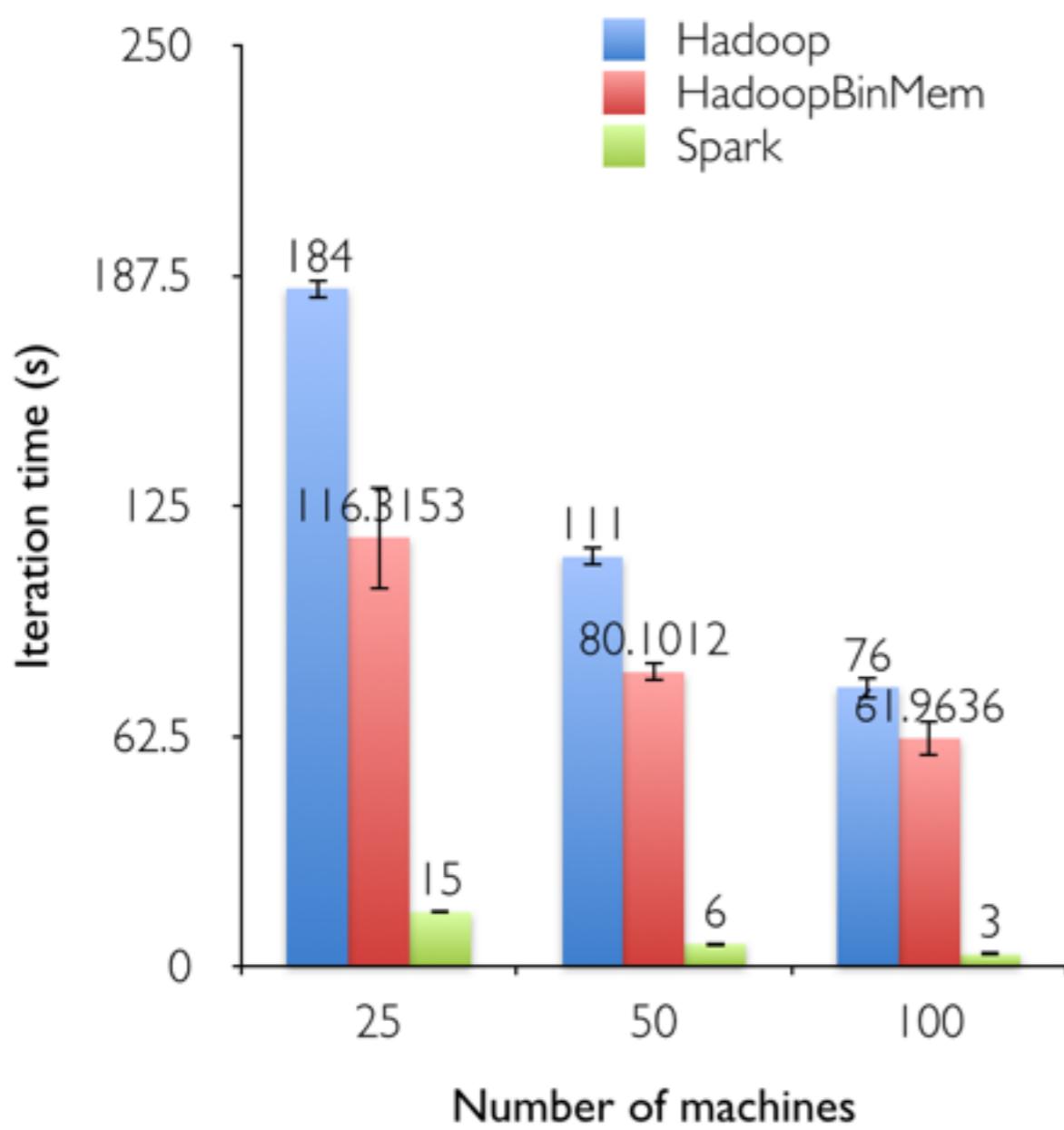
sc.stop()
```

VS Hadoop

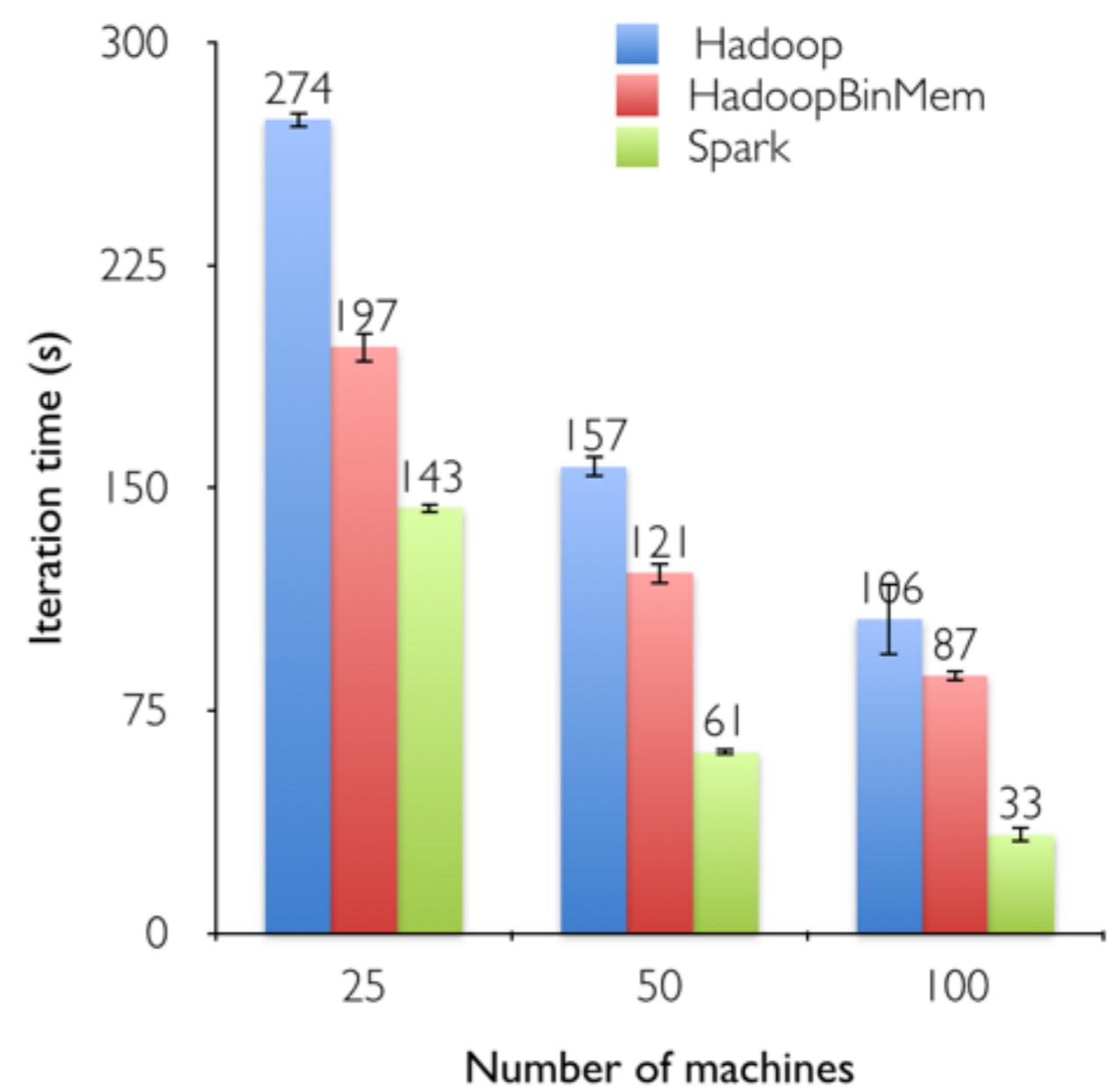


VS Hadoop

Logistic Regression



K-Means



Conclusion

일단 정말이다!

시스템과 분석 조건에 따라 결과가 다를 수 있는 건 본인의 책임이 아니다

Exercises

Environment

- Amazon AWS Account
- Get Ubuntu 14.04 LTS Instance
- MongoDB Install on Ubuntu with apt-get
- JDK Install: sudo apt-get install openjdk-7-jre
- Spark Install on Ubuntu
- Install MongoDB Hadoop Connector with Spark
 - <https://github.com/mongodb/mongo-hadoop/archive/r1.4.2.tar.gz>

AWS Ubuntu configure

- sudo apt-get update
- sudo apt-get install language-pack-ko
- sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
- echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list
- sudo apt-get update
- sudo apt-get install -y mongodb-org

MongoDB

Master/Slave

> mongod --help

Master/slave options (old; use replica sets instead):

- master master mode
- slave slave mode
- source arg when slave: specify master as <server:port>
- only arg when slave: specify a single database to replicate
- slavedelay arg specify delay (in seconds) to be used when applying master ops to slave
- autoresync automatically resync if slave data is stale

Replica set options:

- replSet arg arg is <setname>[<optionalseedhostlist>]
- repIndexPrefetch arg specify index prefetching behavior (if secondary)

Sharding options:

- configsvr declare this is a config db of a cluster; default port 27019; default dir /data/configdb
- shardsvr declare this is a shard db of a cluster; default port 27018

Master 구성

- mongod --dbpath {master_path} --master --port {port}
- mongo --port {port}
 - > show dbs
 - local 0.328125GB
 - > db.printReplicationInfo()
configured oplog size: 192MB
log length start to end: 317secs (0.09hrs)
oplog first event time: Sat Jan 27 2016 11:50:16 GMT+0900 (KST)
oplog last event time: Sat Jan 27 2016 11:55:33 GMT+0900 (KST)
now: Sat Jan 27 2016 11:55:36 GMT+0900 (KST)

Master 구성

- > db.printSlaveReplicationInfo()
local.sources is empty; is this db a --slave?
- > db.printShardingStatus()
printShardingStatus: this db does not have sharding enabled. be sure you are connecting to a mongos from the shell and not to a mongod.

Slave 구성

- mongod --dbpath {slave_path} --slave --port {port} --source {master_ip:maseter_port}

Sat Jan 27 12:04:55.853 [initandlisten] MongoDB starting :
pid=70927 port=48000 dbpath=mongodb_2.4.5/slave slave=1 64-bit
host=nulpulum-mac-13-retina.local

Sat Jan 27 12:04:55.853 [initandlisten] options: { dbpath: "/mongodb_2.4.5/slave", port: 48000, slave: true, source: "localhost:38000" }

Sat Jan 27 12:04:56.891 [replslave] build index done. scanned 0 total records. 0 secs

Master-Slave 확인

- > db.printSlaveReplicationInfo()
=Slave=
source: localhost:38000
syncedTo: Sat Jan 27 2016 12:09:53 GMT+0900 (KST) = 74 secs ago (0.02hrs)
=Master=
local.sources is empty; is this db a – slave?
- On Master
> use SomeDB
> db.data.save({name: 'jungwon', age: undefined});
- On Slave
> use SomeDB
> db.data.find();
{_id: '51f33a81dbed34cf65d102d0', name: 'jungwon', age: undefined}

3-Way Replica Set

3 Instances

- 3개의 동일 Repl 이름을 가진 mongod instance 생성
- mongod --repSet {set_name} --port {each_port} --dbpath {each_path}

Sat Jan 27 14:11:58.007 [initandlisten] MongoDB starting :
pid=71010 port=20000 dbpath=/mongodb_2.4.5/db1 64-bit
host=nulpulum-mac-13-retina.local

Sat Jan 27 14:11:58.299 [websvr] admin web console waiting for
connections on port 21000

Sat Jan 27 14:11:58.300 [initandlisten] waiting for connections on
port 20000

Sat Jan 27 14:12:08.302 [rsStart] replSet can't get
local.system.replset config from self or any seed (EMPTYCONFIG)

Configuration

- mongo {db_ip}:{db_port}
- 동일한 이름 하에 3개의 멤버를 가지는 Repl Set config 생성
> var config={_id:'repl_name', members:[{_id:0, host:'localhost:20000'}, {_id:1, host:'localhost:30000'}, {_id:2, host:'localhost:40000'}] };
- 해당 config를 이용한 rs 셋팅
rs.initiate(config);
{
 "info" : "Config now saved locally. Should come online in about a minute.",
 "ok" : 1
}
• rs.slaveOk()

> rs.help()

rs.status()	{ replSetGetStatus : 1 } checks repl set status
rs.initiate()	{ replSetInitiate : null } initiates set with default settings
rs.initiate(cfg)	{ replSetInitiate : cfg } initiates set with configuration cfg
rs.conf()	get the current configuration object from local.system.replset
rs.reconfig(cfg)	updates the configuration of a running replica set with cfg
rs.add(hostportstr)	add a new member to the set with default attributes
rs.add(membercfgobj)	add a new member to the set with extra attributes
rs.addArb(hostportstr)	add a new member which is arbiterOnly:true
rs.stepDown([secs])	step down as primary (momentarily) (disconnects)
rs.syncFrom(hostportstr)	make a secondary to sync from the given member
rs.freeze(secs)	make a node ineligible to become primary for the time specified
rs.remove(hostportstr)	remove a host from the replica set (disconnects)
rs.slaveOk()	shorthand for db.getMongo().setSlaveOk()
db.isMaster()	check who is primary
db.printReplicationInfo()	check oplog size and time range

Key Point

- Insert는 Primary에서만 가능
- Secondary의 read 시 rs.slaveOk() 필요
- Primary connection fail 시 auto election

N-Way Shard

n Instances

- mongod --shardsvr –dbpath {each_path} --port {each_port}

Sat Jan 27 15:15:40.854 [initandlisten] MongoDB starting :
pid=71105 port=10000 dbpath=/mongodb_2.4.5/db11 64-bit
host=nulpulum-mac-13-retina.local

Sat Jan 27 15:15:41.141 [initandlisten] command local.\$cmd
command: { create: "startup_log", size: 10485760, capped:
true } ntoreturn:1 keyUpdates:0 reslen:37 271ms

Sat Jan 27 15:15:41.141 [websvr] admin web console
waiting for connections on port 11000

Sat Jan 27 15:15:41.142 [initandlisten] waiting for
connections on port 10000

Configure Config Server

- mongod --configsvr --dbpath {config_path} --port {config_port}

Sat Jan 27 15:22:12.404 [initandlisten] MongoDB starting :
pid=71141 port=40000 dbpath=/mongodb_2.4.5/config1
master=1 64-bit host=nulpulum-mac-13-retina.local

Sat Jan 27 15:22:12.544 [initandlisten] creating replication
oplog of size: 5MB...

Sat Jan 27 15:22:12.547 [initandlisten] *****

Sat Jan 27 15:22:12.547 [websvr] admin web console waiting
for connections on port 41000

Sat Jan 27 15:22:12.547 [initandlisten] waiting for connections
on port 40000

> mongos --help

- Sharding options:

--configdb arg 1 or 3 comma separated config servers

--localThreshold arg ping time (in ms) for a node to be considered local (default 15ms)

--test just run unit tests

--upgrade upgrade meta data version

--chunkSize arg maximum amount of data per chunk (기본 : 64kbytes)

--ipv6 enable IPv6 support (disabled by default)

--jsonp allow JSONP access via http (has security implications)

--noscripting disable scripting engine

Mongo Server

- mongos --configdb {cfg_ip:cfg_port} --chunkSize 1 --port {server_port}
- mongo –port {mongos_port}
- > use admin
- > db.runCommand({addShard: '{shard_ip:shard:port}'});
 { "shardAdded" : "shard0000", "ok" : 1 }
- > db.runCommand({ enablesharding: 'db_name' });
- db.runCommand({ shardcollection: 'db_name.collection_name', key:{_id:1} });

Shard Test

- On mongos
- > use {db_name}
- ```
for(var i=0; i < 10000000 ; i++)
{ db.db_name.save({ age: i, name:i+", address: i+'street'}); };
```

# **Spark with MongoDB**

# Environment

- Install Spark
  - <http://www.apache.org/dyn/closer.lua/spark/spark-1.6.0/spark-1.6.0.tgz>
- Install MongoDB
- Install JAVA: sudo apt-get install default-jdk
- Install MongoDB Hadoop Connector
- Download Sample Data

# Sort

```
from __future__ import print_function
import sys
from pyspark import SparkContext
if __name__ == "__main__":
 if len(sys.argv) != 2:
 print("Usage: sort <file>", file=sys.stderr)
 exit(-1)
 sc = SparkContext(appName="PythonSort")
 lines = sc.textFile(sys.argv[1], 1)
 sortedCount = lines.flatMap(lambda x: x.split(' ')) \
 .map(lambda x: (int(x), 1)) \
 .sortByKey(lambda x: x)
 output = sortedCount.collect()
 for (num, unitcount) in output:
 print(num)
 sc.stop()

spark-submit sort.py {target_file}
```

# WordCount

```
from __future__ import print_function
import sys
from operator import add
from pyspark import SparkContext

if __name__ == "__main__":
 if len(sys.argv) != 2:
 print("Usage: wordcount <file>", file=sys.stderr)
 exit(-1)
 sc = SparkContext(appName="PythonWordCount")
 lines = sc.textFile(sys.argv[1], 1)
 counts = lines.flatMap(lambda x: x.split(' '))
 .map(lambda x: (x, 1))
 .reduceByKey(add)
 output = counts.collect()
 for (word, count) in output:
 print("%s: %i" % (word, count))

 sc.stop()

spark-submit wordcount.py {target_file}
```

# Configure MongoDB

- Start MongoDB

```
mongod -f /etc/mongod.conf
```

- Load Sample Data

```
mongoimport mstf.csv --type csv --headerline -d marketdata -c minibars –port {db_port}
```

- > use marketdata

- > db.minbars.findOne()

# Test with python

- Data를 Spark RDD format으로 변환

```
config = {"mongo.input.uri": "mongodb://localhost:27017/
marketdata.minbars"}
```

```
inputFormatClassName = "com.mongodb.hadoop.MongoInputFormat"
```

```
keyClassName = "org.apache.hadoop.io.Text"
```

```
valueClassName = "org.apache.hadoop.io.MapWritable"
```

```
minBarRawRDD = sc.newAPIHadoopRDD(inputFormatClassName,
keyClassName, valueClassName, None, None, config)
```

# Test with python

- output을 MongoDB에 저장

```
config["mongo.output.uri"] = "mongodb://localhost:
27017/marketdata.fiveminutebars"
```

```
outputFormatClassName =
"com.mongodb.hadoop.MongoOutputFormat"
```

- Raw 데이터 치환

```
minBarRDD = minBarRawRDD.values()
```

# Test with python

- 시간순 정렬 & 5분 단위 group

```
import calendar, time, math
```

```
dateFormatString = '%Y-%m-%d %H:%M'
```

```
groupedBars = minBarRDD.sortBy(lambda doc:
str(doc["Timestamp"])).groupBy(lambda doc: (doc["Symbol"],
math.floor(calendar.timegm(time.strptime(doc["Timestamp"],
dateFormatString)) / (5*60))))
```

# Test with python

```
def ohlc(grouping):
 low = sys.maxint
 high = -sys.maxint
 i = 0
 groupKey = grouping[0]
 group = grouping[1]
 for doc in group:
 if i == 0:
 openTime = doc["Timestamp"]
 openPrice = doc["Open"]
 if doc["Low"] < low:
 low = doc["Low"]
```

# Test with python

```
if doc["High"] > high:
 high = doc["High"]

i = i + 1
take close of last bar
if i == len(group):
 close = doc["Close"]
 outputDoc = {"Symbol": groupKey[0],
"Timestamp": openTime,
"Open": openPrice,
"High": high,
"Low": low,
"Close": close}
return (None, outputDoc)

resultRDD = groupedBars.map(ohlc)

resultRDD.saveAsNewAPIHadoopFile("file:///placeholder", //MongoDB에 output 저장
outputFormatClassName, None, None, None, None, config)
```

<https://github.com/nicejwjin/MongoDB-Spark.git>

감사합니다

[meteorstartup@gmail.com](mailto:meteorstartup@gmail.com)