#madmug

MongoDB + Hadoop

Norberto Leite

Wingman, MongoDB Inc.

mongoDB

Agenda

- MongoDB
- Hadoop
- MongoDB + Hadoop Connector
- How it works
- What can we do with it

MongoDB

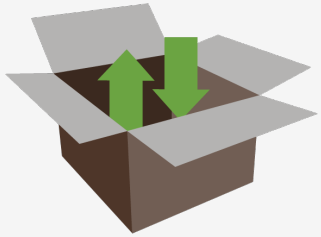
mongoDB

MongoDB

The leading NoSQL database

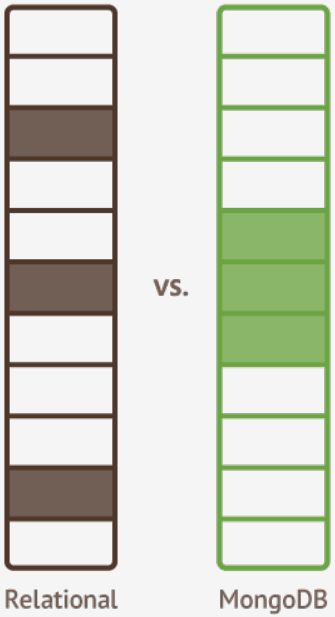
General Purpose

Document Database

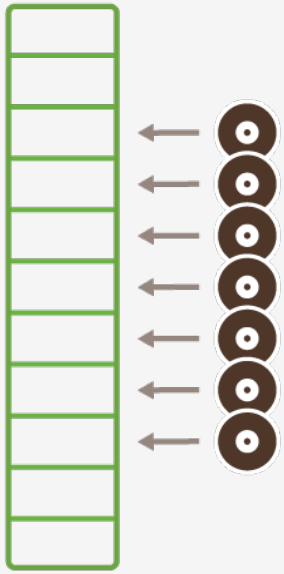


Open-Source

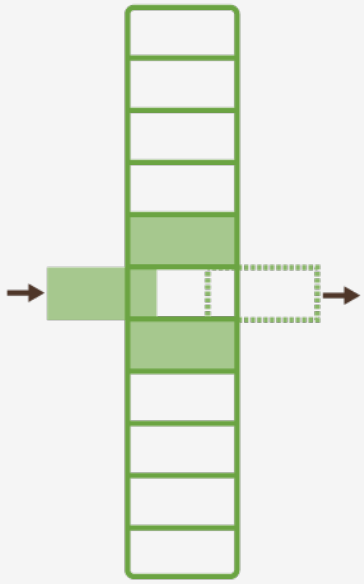
Performance



Better Data Locality

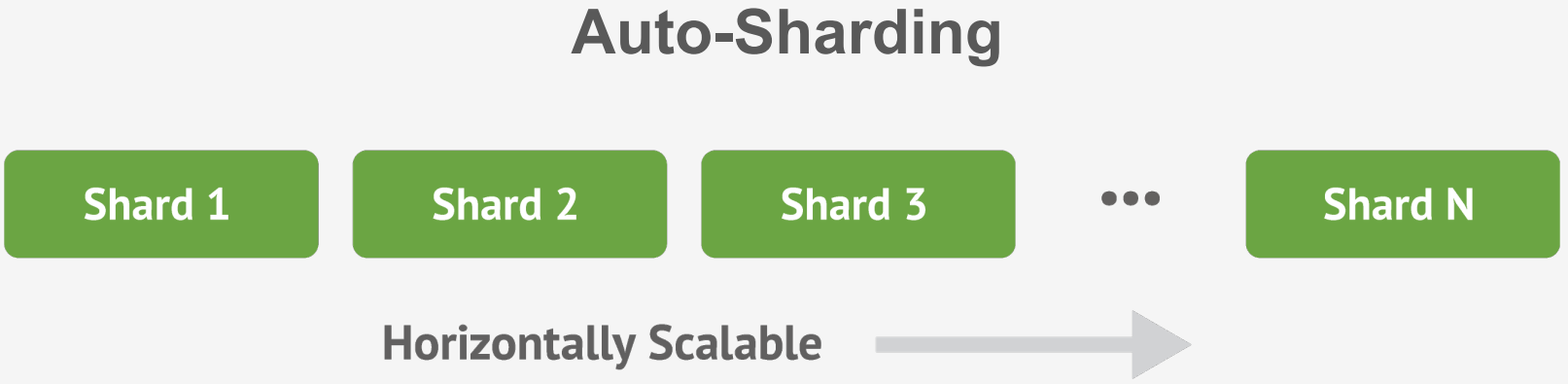


In-Memory Caching



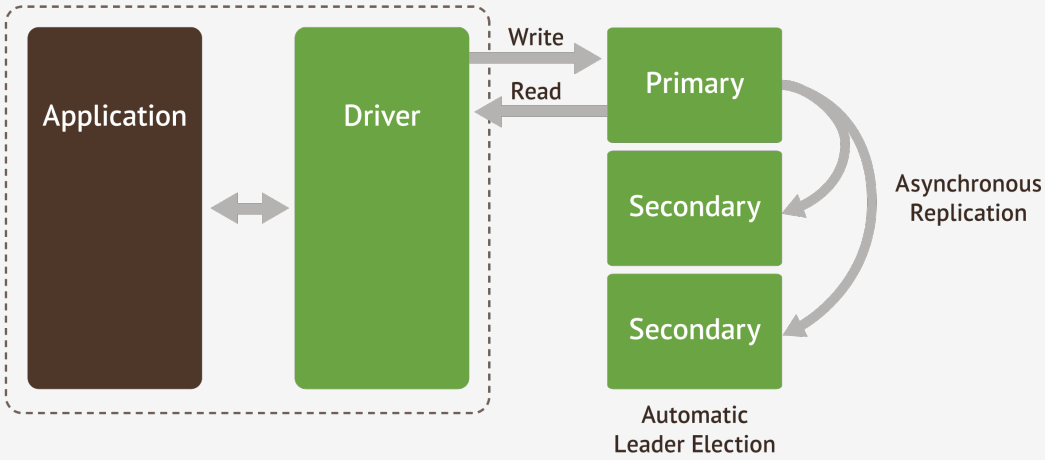
In-Place Updates

Scalability



- Increase capacity as you go
- Commodity and cloud architectures
- Improved operational simplicity and cost visibility

High Availability



- Automated replication and failover
- Multi-data center support
- Improved operational simplicity (e.g., HW swaps)
- Data durability and consistency

Shell and Drivers

Drivers

Drivers for most popular programming languages and frameworks


Java


Ruby

JavaScript


Perl


Python


Haskell

Shell

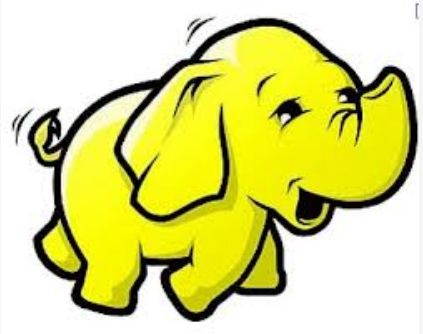
Command-line shell for interacting directly with database

```
> db.collection.insert({product:"MongoDB",
type:"Document Database"})
>
> db.collection.findOne()
{
        "_id"           : ObjectId("5106c1c2fc629bfe52792e86"),
        "product"       : "MongoDB"
        "type"          : "Document Database"
}
```

Hadoop

mongoDB

Hadoop



- Google publishes seminal papers
  - GFS – global file system (Oct 2003)
  - MapReduce – divide and conquer (Dec 2004)
  - How they indexed the internet
- Yahoo builds and open sources (2006)
  - Doug Cutting lead, now at Cloudera
  - Most others now at Hortonworks
- Commonly mentioned has:
  - The elephant in the room!

Hadoop

- Primary components
  - HDFS – Hadoop Distributed File System
  - MapReduce – parallel processing engine
- Ecosystem
  - HIVE
  - HBASE
  - PIG
  - Oozie
  - Sqoop
  - Zookeeper

Apache Hadoop Ecosystem

Ambari
Provisioning, Managing and Monitoring Hadoop Clusters

Sqoop
Data Exchange

Flume
Log Collector

Zookeeper
Coordination

Oozie
Workflow

Pig
Scripting

Mahout
Machine Learning

R Connectors
Statistics

Hive
SQL Query

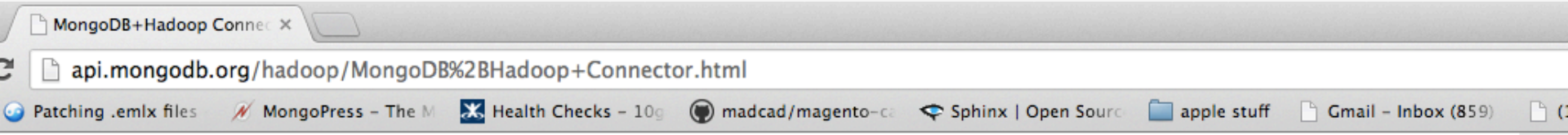
Hbase
Columnar Store

YARN Map Reduce v2
Distributed Processing Framework

HDFS
Hadoop Distributed File System

MongoDB+Hadoop Connector

mongoDB



MongoDB+Hadoop Connector

CURRENT RELEASE: 1.0.0

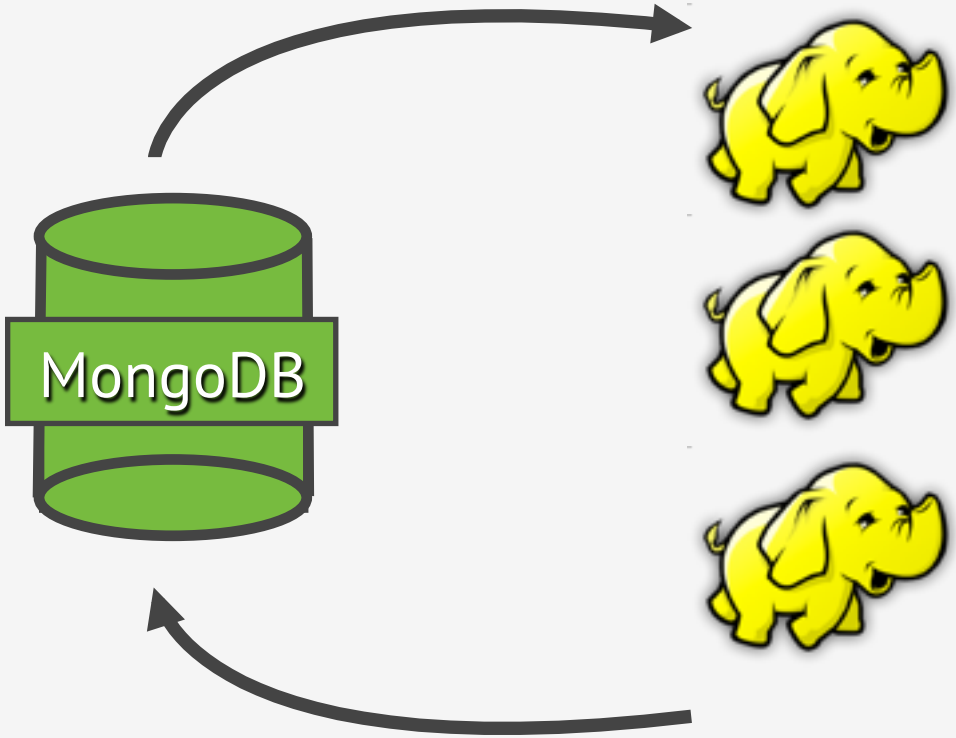
The *Mongo+Hadoop Connector* (for brevitys sake, we'll refer to it as *mongo-hadoop* in this documentation) is a series of plugins for the Apache Hadoop Platform to allow connectivity to MongoDB. This connectivity takes the form of allowing both reading MongoDB data into Hadoop (for use in MapReduce jobs as well as other components of the Hadoop ecosystem), as well as writing the results of Hadoop jobs out to MongoDB. A forthcoming release will also allow for reading and writing static BSON files (ala *mongodump / mongorestore*) to allow offline batching; commonly, users find this to be a beneficial feature to run analytics against backup data.

At this time, we support the "core" Hadoop APIs (now known as Hadoop Common), in the form of *mongo-hadoop-core*. There is additionally support for other pieces of the Hadoop Ecosystem, including Pig for ETL and Streaming for running Mongo+Hadoop jobs with Python (future releases will support additional scripting languages such as Ruby). Although it is not dependent upon Hadoop, we also provide a connector for the Flume distributed logging system.

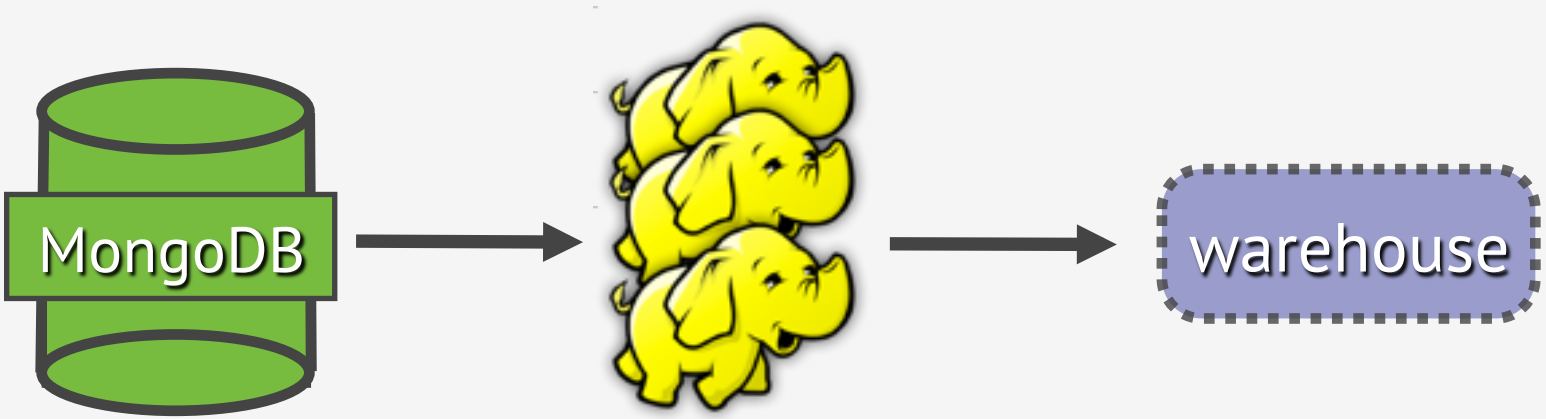
MongoDB Hadoop Connector

- http://api.mongodb.org/hadoop/MongoDB%2BHadoop+Connector.html
- Allows interoperation of MongoDB and Hadoop
  - "Give power to the people"
- Allows processing across multiple sources
- Avoid custom hacky exports and imports
- Scalability and Flexibility to accommodate Hadoop and|or MongoDB changes

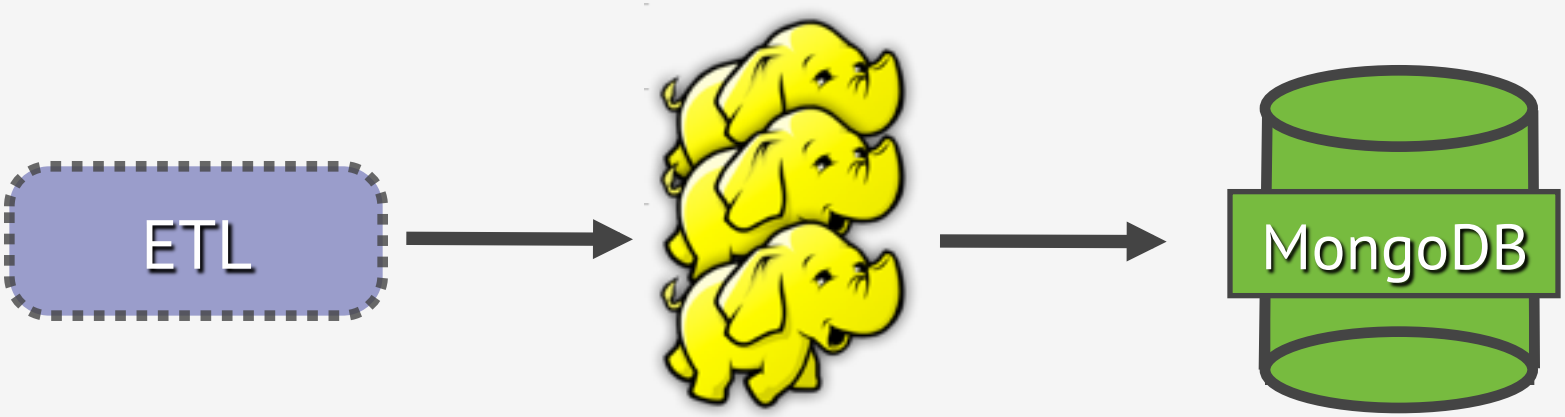
MongoDB with Hadoop



MongoDB with Hadoop



MongoDB with Hadoop



Benefits and Features

- Full multi-core parallelism to process data in MongoDB
- Full integration w/ Hadoop and JVM ecosystem
- Can be used on Amazon Elastic MapReduce
- Read and write backup files to local, HDFS and S3
- Vanilla Java MapReduce
  - But not only using Hadoop Streaming



Benefits and Features

- Supports PIG

- Supports HIVE

How it works

- Adapter examines MongoDB input collection and calculates a set of **splits** from data
- Each split is assigned to a Hadoop node
- In parallel hadoop pulls data from splits on MongoDB (or BSON) and starts processing locally
- Hadoop merges results and streams output back to MongoDB (or BSON) output collection

Example Tour

mongoDB

Tour bus stops

- Java MapReduce w/ MongoDB-Hadoop Connector
- Using Hadoop Streaming
- Pig and Hive

Data Set

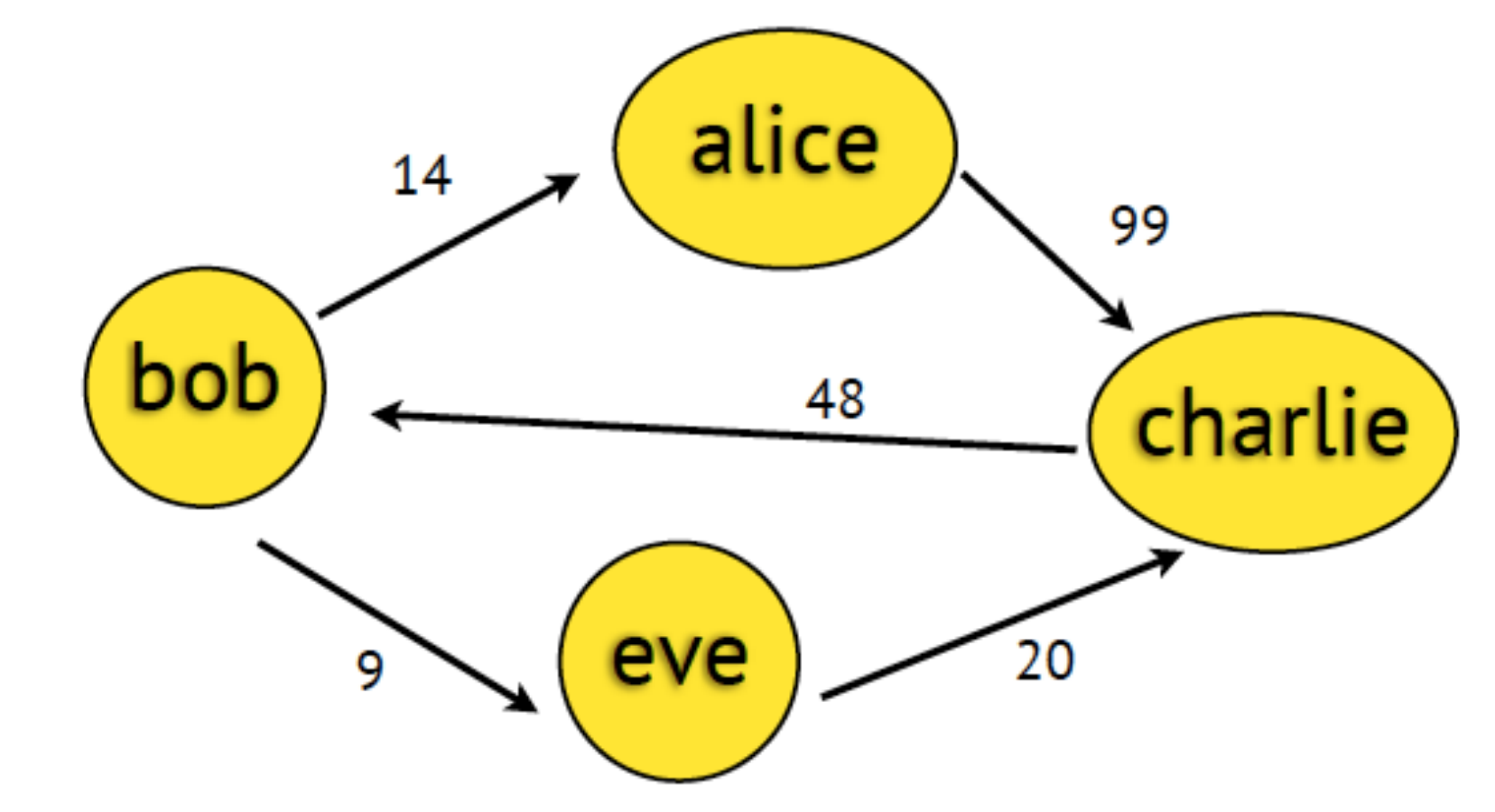
- ENRON emails corpus (501 records, 1.75GB)
- Each document is one email
- https://www.cs.cmu.edu/~enron/

Document Example


```
{
"_id" : ObjectId("4f2ad4c4d1e2d3f15a000000"),
"body" : "Here is our forecast\n\n ",
"filename" : "1.",
"headers" : {
      "From" : "phillip.allen@enron.com",
      "Subject" : "Forecast Info",
      "X-bcc" : "",
      "To" : "tim.belden@enron.com",
      "X-Origin" : "Allen-P",
      "X-From" : "Phillip K Allen",
      "Date" : "Mon, 14 May 2001 16:39:00 -0700 (PDT)",
      "X-To" : "Tim Belden ",
      "Message-ID" : "<18782981.1075855378110.JavaMail.evans@thyme>",
      "Content-Type" : "text/plain; charset=us-ascii",
      "Mime-Version" : "1.0"
      }
}
```

Let's build a graph between senders and recipients and count the messages exchanged

Graph Sketch



Receiver Sender Pairs

{"_id": {"t":"bob@enron.com", "f":"alice@enron.com"}, "count" : 14}

{"_id": {"t":"bob@enron.com", "f":"eve@enron.com"}, "count" : 9}

{"_id": {"t":"alice@enron.com", "f":"charlie@enron.com"}, "count" : 99}

{"_id": {"t":"charlie@enron.com", "f":"bob@enron.com"}, "count" : 48}

{"_id": {"t":"eve@enron.com", "f":"charlie@enron.com"}, "count" : 20}

Vanilla Java MapReduce

mongoDB

Map Phase – each document get's through mapper function

```
@Override
public void map(NullWritable key, BSONObject val, final Context context){
BSONObject headers = (BSONObject)val.get("headers");
if(headers.containsKey("From") && headers.containsKey("To")){
String from = (String)headers.get("From"); String to = (String) headers.get("To");
String[] recips = to.split(",");
for(int i=0;i<recips.length;i++){
    String recip = recips[i].trim();
    context.write(new MailPair(from, recip), new IntWritable(1));
    }
}
}
```

Reduce Phase – output Maps are grouped by key and passed to Reducer

```
public void reduce( final MailPair pKey,  final Iterable<IntWritable> pValues,
final Context pContext ){
int sum = 0;
for ( final IntWritable value : pValues ){
    sum += value.get();
}
BSONObject outDoc = new BasicDBObjectBuilder().start()
    .add( "f" , pKey.from)
    .add( "t" , pKey.to )
    .get();
BSONWritable pkeyOut = new BSONWritable(outDoc);
pContext.write( pkeyOut, new IntWritable(sum) ); }
```

Read From MongoDB (or BSON)

mongo.job.input.format=com.mongodb.hadoop.MongoInputFormat

mongo.input.uri=mongodb://my-db:27017/enron.messages

mongo.job.input.format=com.mongodb.hadoop.BSONFileInputFormat

mapred.input.dir= file:///tmp/messages.bson

mapred.input.dir= hdfs:///tmp/messages.bson

mapred.input.dir= s3:///tmp/messages.bson

Write To MongoDB (or BSON)

mongo.job.output.format=com.mongodb.hadoop.MongoOutputFormat

mongo.output.uri=mongodb://my-db:27017/enron.results_out

mongo.job.output.format=com.mongodb.hadoop.BSONFileOutputFormat

mapred.output.dir= file:///tmp/results.bson

mapred.output.dir= hdfs:///tmp/results.bson

mapred.output.dir= s3:///tmp/results.bson

Query Data

```
mongos> db.streaming.output.find({"_id.t": /^kenneth.lay/})
{ "_id" : { "t" : "kenneth.lay@enron.com",
        "f" : "15126-1267@m2.innovyx.com" }, "count" : 1 }
{ "_id" : { "t" : "kenneth.lay@enron.com",
        "f" : "2586207@www4.imakenews.com" }, "count" : 1 }
{ "_id" : { "t" : "kenneth.lay@enron.com",
        "f" : "40enron@enron.com" }, "count" : 2 }
{ "_id" : { "t" : "kenneth.lay@enron.com",
        "f" : "a..davis@enron.com" }, "count" : 2 }
{ "_id" : { "t" : "kenneth.lay@enron.com",
        "f" : "a..hughes@enron.com" }, "count" : 4 }
{ "_id" : { "t" : "kenneth.lay@enron.com",
        "f" : "a..lindholm@enron.com" }, "count" : 1 }
{ "_id" : { "t" : "kenneth.lay@enron.com",
        "f" : "a..schroeder@enron.com" }, "count" : 1 }
```

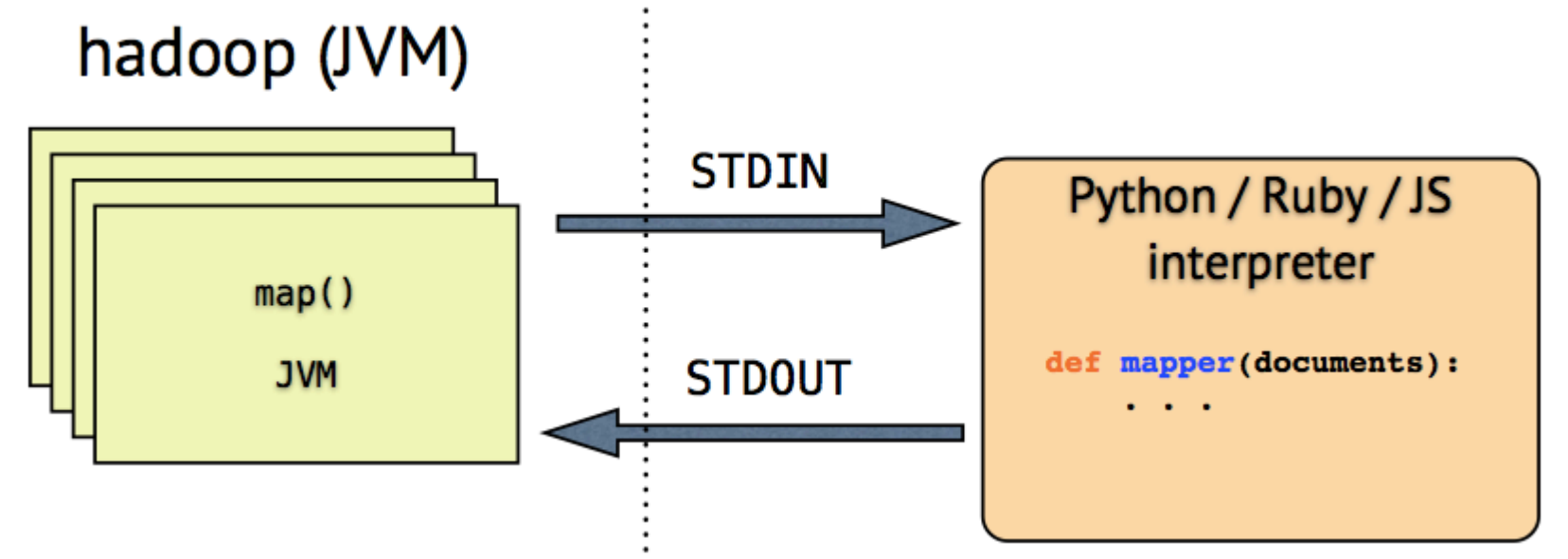
Hadoop Streaming

mongoDB

Same thing but lets use Python

PYTHON FTW!!!

Hadoop Streaming: How it works



Install python library

```
> pip install pymongo_hadoop
```

Map Phase

```
from pymongo_hadoop import BSONMapper
def mapper(documents): i=0
for doc in documents: i=i+1
        from_field  = doc['headers']['From']
        to_field = doc['headers']['To']
        recips = [x.strip() for x in to_field.split(',')]
        for r in recips:
            yield {'_id': {'f':from_field, 't':r}, 'count': 1}
BSONMapper(mapper)
print >> sys.stderr, "Done Mapping."
```

Reduce Phase

```
from pymongo_hadoop import BSONReducer
def reducer(key, values):
    print >> sys.stderr, "Processing from/to %s" % str(key)
    _count = 0
    for v in values:
        _count += v['count']
    return {'_id': key, 'count': _count}
BSONReducer(reducer)
```

Surviving Hadoop

mongoDB

MapReduce easier w/ PIG and Hive

- PIG
  - Powerful language
  - Generates sophisticated map/reduce
  - Workflows from simple scripts
- HIVE
  - Similar to PIG
  - SQL as language

PIG

mongoDB

MongoDB+Hadoop and PIG

```
data = LOAD 'mongodb://localhost:27017/db.collection'
    using com.mongodb.hadoop.pig.MongoLoader;


STORE records INTO 'file:///output.bson'
     using com.mongodb.hadoop.pig.BSONStorage;
```

MongoDB + Hadoop and PIG

- PIG has a some special datatypes
  - Bags
  - Maps
  - Tuples
- MongoDB+Hadoop Connector converts between PIG and MongoDB datatypes

PIG

```
raw = LOAD 'hdfs:///messages.bson'
    using com.mongodb.hadoop.pig.BSONLoader('','headers:[]') ;
send_recip = FOREACH raw GENERATE $0#'From' as from, $0#'To'
as to;
//filter && split
send_recip_filtered = FILTER send_recip BY to IS NOT NULL;
send_recip_split = FOREACH send_recip_filtered GENERATE
    from as from, TRIM(FLATTEN(TOKENIZE(to))) as to;
//group && count
send_recip_grouped = GROUP send_recip_split BY (from, to);
send_recip_counted = FOREACH send_recip_grouped GENERATE
    group, COUNT($1) as count;
STORE send_recip_counted INTO 'file:///enron_results.bson' using
com.mongodb.hadoop.pig.BSONStorage;
```

HIVE

mongoDB

Hive w/ MongoDB-Hadoop

- Similar to Pig
  - Process data without need to write MapReduce from scratch
- But with SQL !

Data set example

```
db.users.find()
{ "_id": 1, "name": "Tom", "age": 28 }
{ "_id": 2, "name": "Alice", "age": 18 }
{ "_id": 3, "name": "Bob", "age": 29 }
{ "_id": 101, "name": "Scott", "age": 10 }
{ "_id": 104, "name": "Jesse", "age": 52 }
{ "_id": 110, "name": "Mike", "age": 32 }
...
```

Hive

```
CREATE TABLE mongo_users (id int, name string, age int)
STORED BY "com.mongodb.hadoop.hive.MongoStorageHandler"
WITH SERDEPROPERTIES( "mongo.columns.mapping" =
"_id,name,age" )
TBLPROPERTIES ( "mongo.uri" = "mongodb://localhost:27017/
test.users");
```

Hive

```
SELECT name,age FROM mongo_users WHERE id > 100 ;
```

```
SELECT * FROM mongo_users GROUP BY age WHERE id > 100 ;
```

```
SELECT * FROM mongo_users T1 JOIN user_emails T2 WHERE T1.id =
T2.id;
```

Hive

```
INSERT OVERWRITE TABLE old_users SELECT id,name,age FROM
mongo_users WHERE age > 100 ;
```

```
DROP TABLE mongo_users;
```

Upcoming

mongoDB

Roadmap Features

- Performance Improvements – Lazy BSON
- Full-Featured Hive Support
- Support multi-collection input
- API for **custom splitter** implementations
- And lots more …

Recap

- Use Hadoop for massive MapReduce computations on big data sets stored in MongoDB
- MongoDB can be used as Hadoop filesystem
- There's lots of tools to make it easier
  - Streaming
  - Hive
  - PIG
  - EMR
- https://github.com/mongodb/mongo-hadoop/tree/master/examples


That's all Folks!

QA?

mongoDB

#madmug

MongoDB + Hadoop

Norberto Leite

Senior Solutions Architect, MongoDB Inc.

mongoDB