Scalable data processing on computer clusters

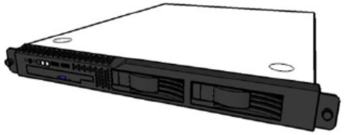
Gergely Lukács
Pázmány Péter Catholic University
Faculty of Information Technology
Budapest, Hungary
lukacs@itk.ppke.hu

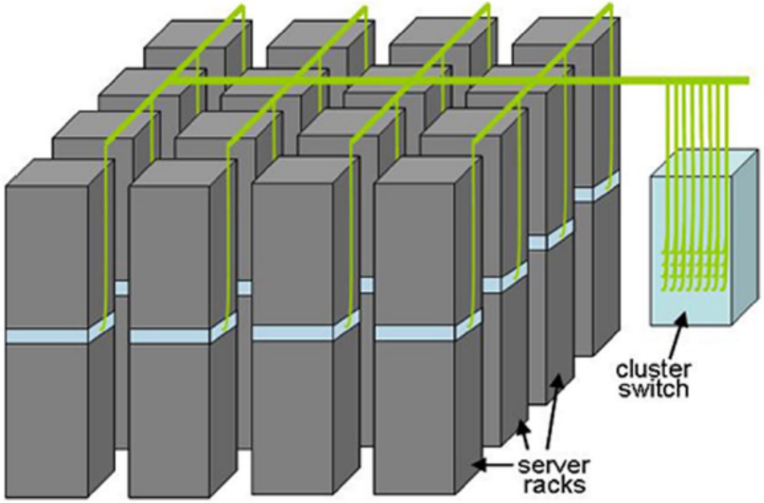
Problem with amount of data?

- Better algorithms
- Numba, C/Cython
- More efficient formats for data
- Data subsampling

- Laptop (4 cores, 16GB RAM, 1 TB hdd)
- Server (24 cores, 1TB RAM)
- Parallelization
  - Methods/architectures for parallelization
- HW accelerators (depending on task: GPGPU, FPGA)

Building Blocks

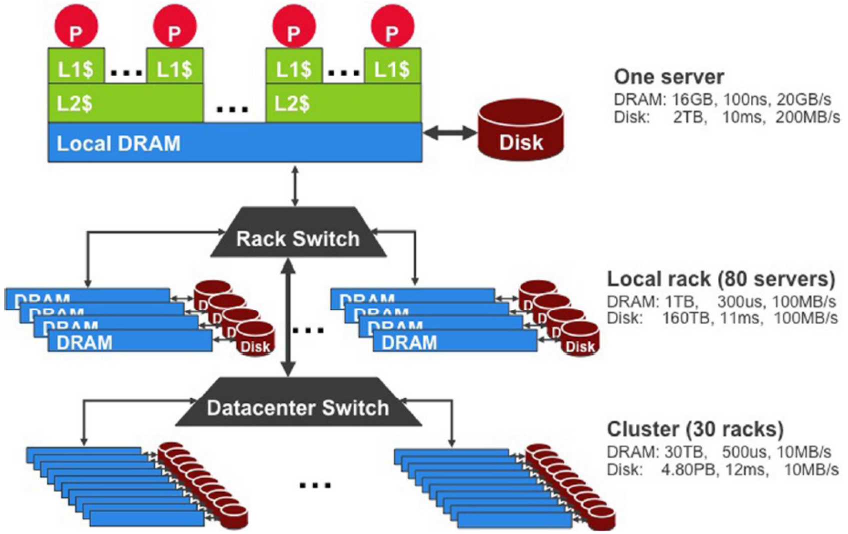


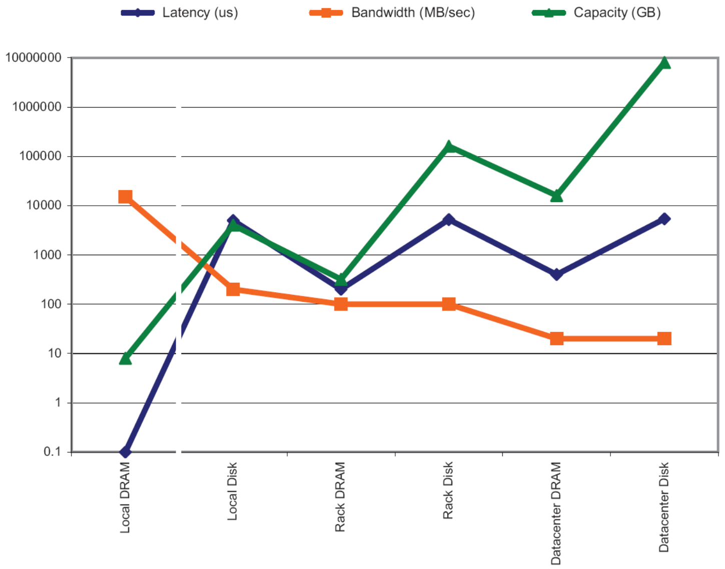


Major ideas

- Process data sequentially, avoid random access
  - Seeks are expensive, disk throughput is reasonable
- Move processing to the data
  - Cluster have limited bandwidth
- Seamless scalability
- From the mythical man-month to the tradable machine-hour
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates

Storage Hierarchy, Seek vs. scan

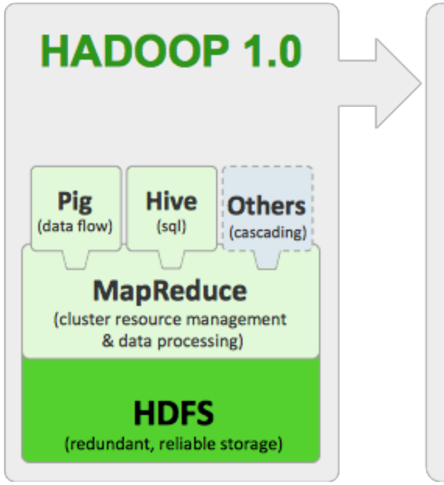




- Consider a 1 TB database with 100 byte records
  - We want to update 1 percent of the records
- Scenario 1: random access
  - Each update takes ~30 ms (seek, read, write)
  - 10^8 updates = ~35 days
- Scenario 2: rewrite all records
  - Assume 100 MB/s throughput
  - Time = 5.6 hours(!)
- Lesson: avoid random seeks!

Hadoop (1.0), MapReduce

- Elosztott fájlrendszer: HDFS
  - Skálázódás, hibatűrés
  - Elosztott, redundáns
- MapReduce



Map/Reduce

- Map/Reduce
  - Programming model from Lisp (and other functional languages)
- Many problems can be phrased this way
- Easy to distribute across nodes
- Nice retry/failure semantics

Map in Lisp (Scheme)

Unary operator

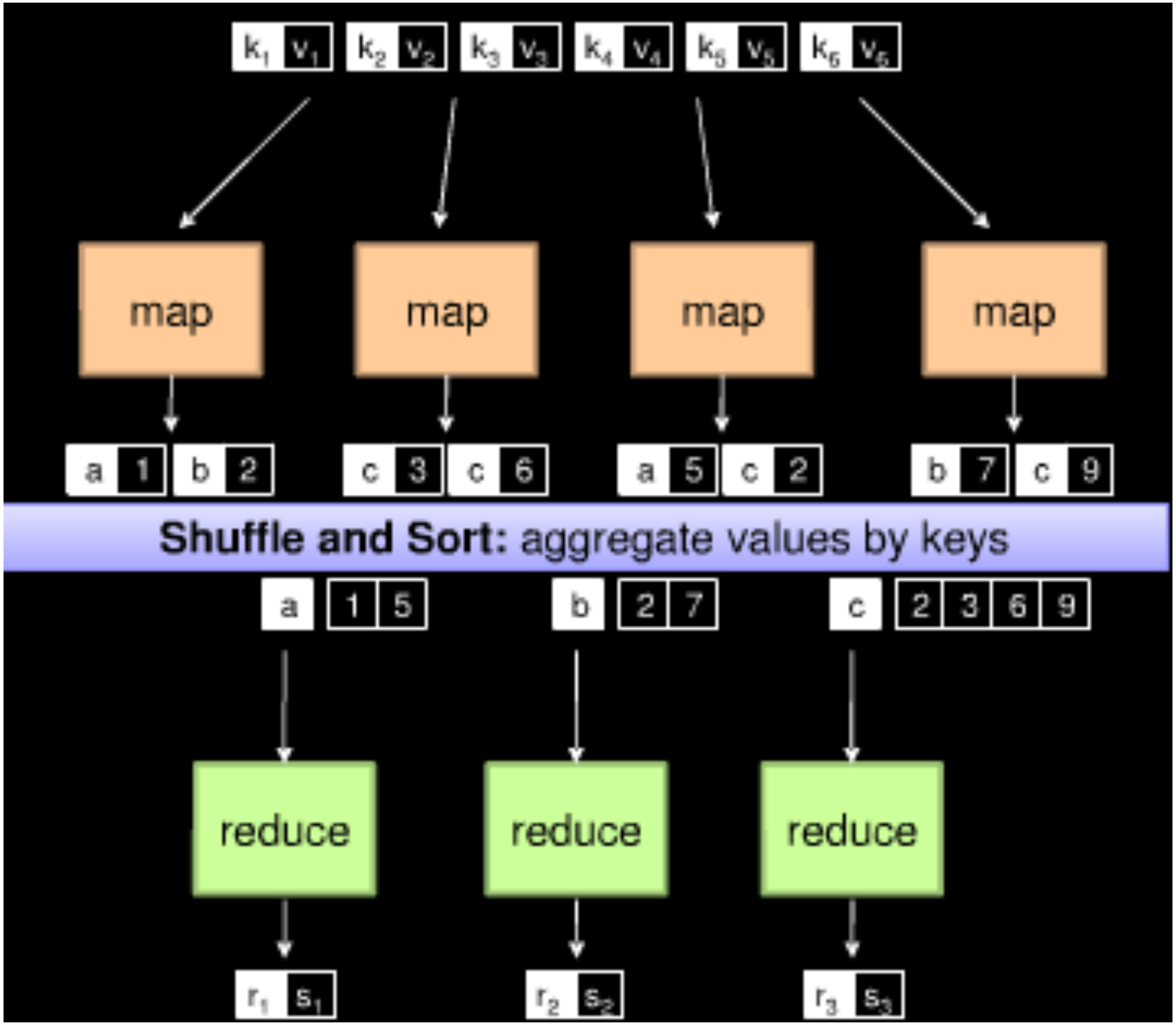
- (map ***f list [list***$_{2}$ ***list***$_{3}$ ***…]***)

Binary operator

- (map square ‘(1 2 3 4))
  - (1 4 9 16)
- (reduce + ‘(1 4 9 16))
  - 30
- (reduce + (map square (map – l$_{1}$ l$_{2}$))))

Map/Reduce ala Google

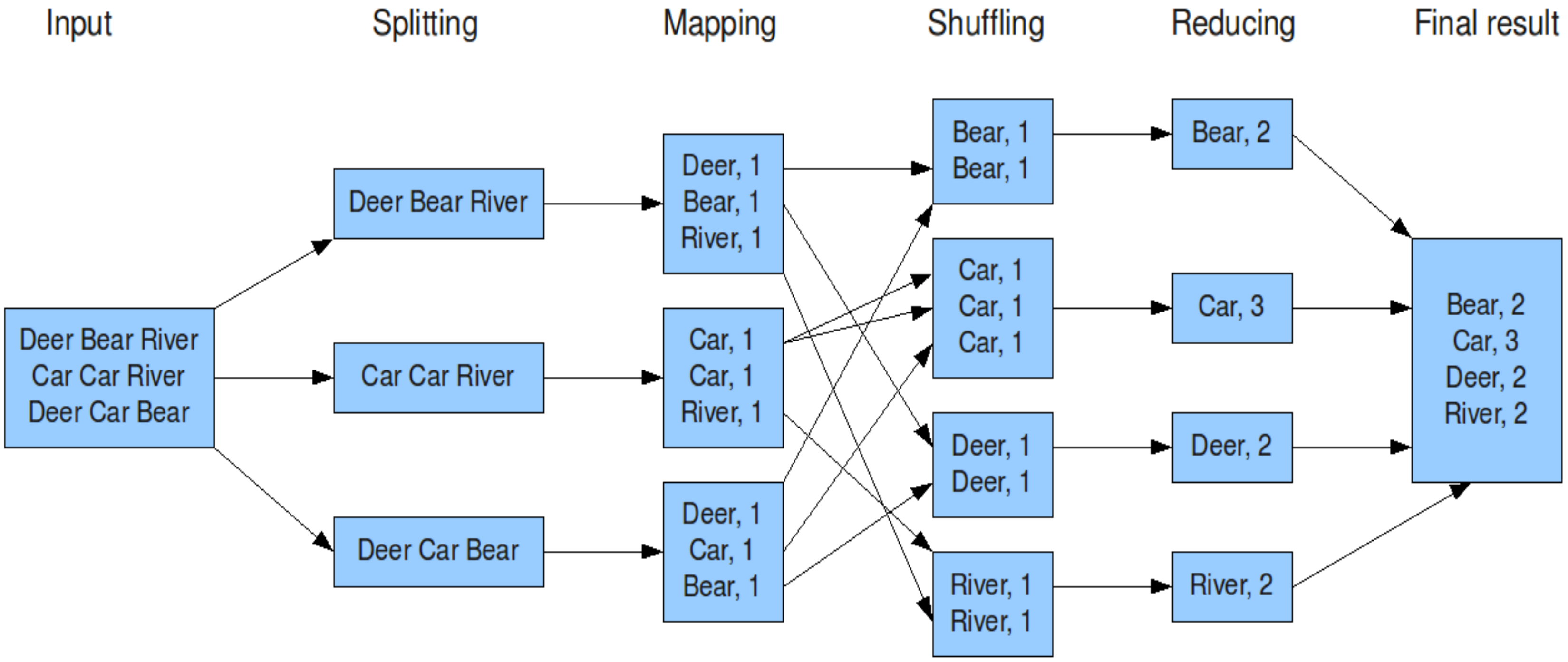
- map(key, val) is run on each item in set
  - emits new-key / new-val pairs
- reduce(key, vals)
  - All values with the same key are reduced together
  - emits final output


k_1 v_1 k_2 v_2 k_3 v_3 k_4 v_4 k_5 v_5 k_6 v_6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 9
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 9
reduce
reduce
reduce
r_1 s_1
r_2 s_2
r_3 s_3

count words in docs

- Input consists of (url, contents) pairs
- map(key=url, val=contents):
  - For each word *w* in contents, emit (w, “1”)
- reduce(key=word, values=uniq_counts):
  - Sum all “1”s in values list
  - Emit result “(word, sum)”

The overall MapReduce word count process



Input

Deer Bear River
Car Car River
Deer Car Bear

Splitting

Deer Bear River

Car Car River

Deer Car Bear

Mapping

Deer, 1
Bear, 1
River, 1

Car, 1
Car, 1
River, 1

Deer, 1
Car, 1
Bear, 1

Shuffling

Bear, 1
Bear, 1

Car, 1
Car, 1
Car, 1

Deer, 1
Deer, 1

River, 1
River, 1

Reducing

Bear, 2

Car, 3

Deer, 2

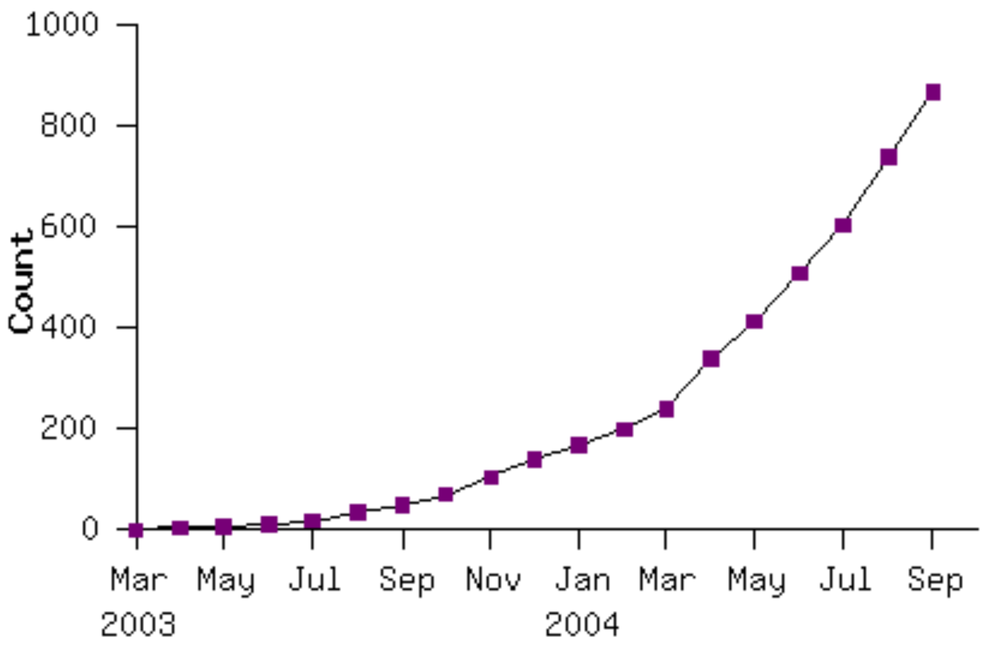
River, 2

Final result

Bear, 2
Car, 3
Deer, 2
River, 2

Model is Widely Applicable

MapReduce Programs In Google Source Tree



Example uses:

distributed grep

term-vector / host

document clustering

...

distributed sort

web access log stats

machine learning

...

web link-graph reversal

inverted index construction

statistical machine translation

...

Execution Framework

- The execution framework handles everything else…
  - Scheduling: assigns workers to map and reduce tasks
  - “Data distribution”: moves processes to data
  - Synchronization: gathers, sorts, and shuffles intermediate data
  - Errors and faults: detects worker failures and restarts
- Limited control over data and execution flow
  - All algorithms must expressed in m, r, c, p
- You don’t know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing

In MapReduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                 Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
             OutputCollector<Text, Text> oc,
                 Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));

                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);

        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
    Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pag
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class)
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.cla
        FileInputFormat.addInputPath(lfu, new
    Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_us
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputForm
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
    Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
    Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
    Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFor
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.cl
        group.setOutputFormat(SequenceFileOutputFormat.class)
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
    Path("/user/gates/tmp/joined"));
       FileOutputFormat.setOutputPath(group, new
    Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.cl
       top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFo
        top100.setOutputKeyClass(LongWritable.cla
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutput
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class
        top100.setReducerClass(LimitClicks.class)
        FileInputFormat.addInputPath(top100, new
    Path("/user/gates/tmp/grouped"));
       FileOutputFormat.setOutputPath(top100, new
    Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for u
    18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

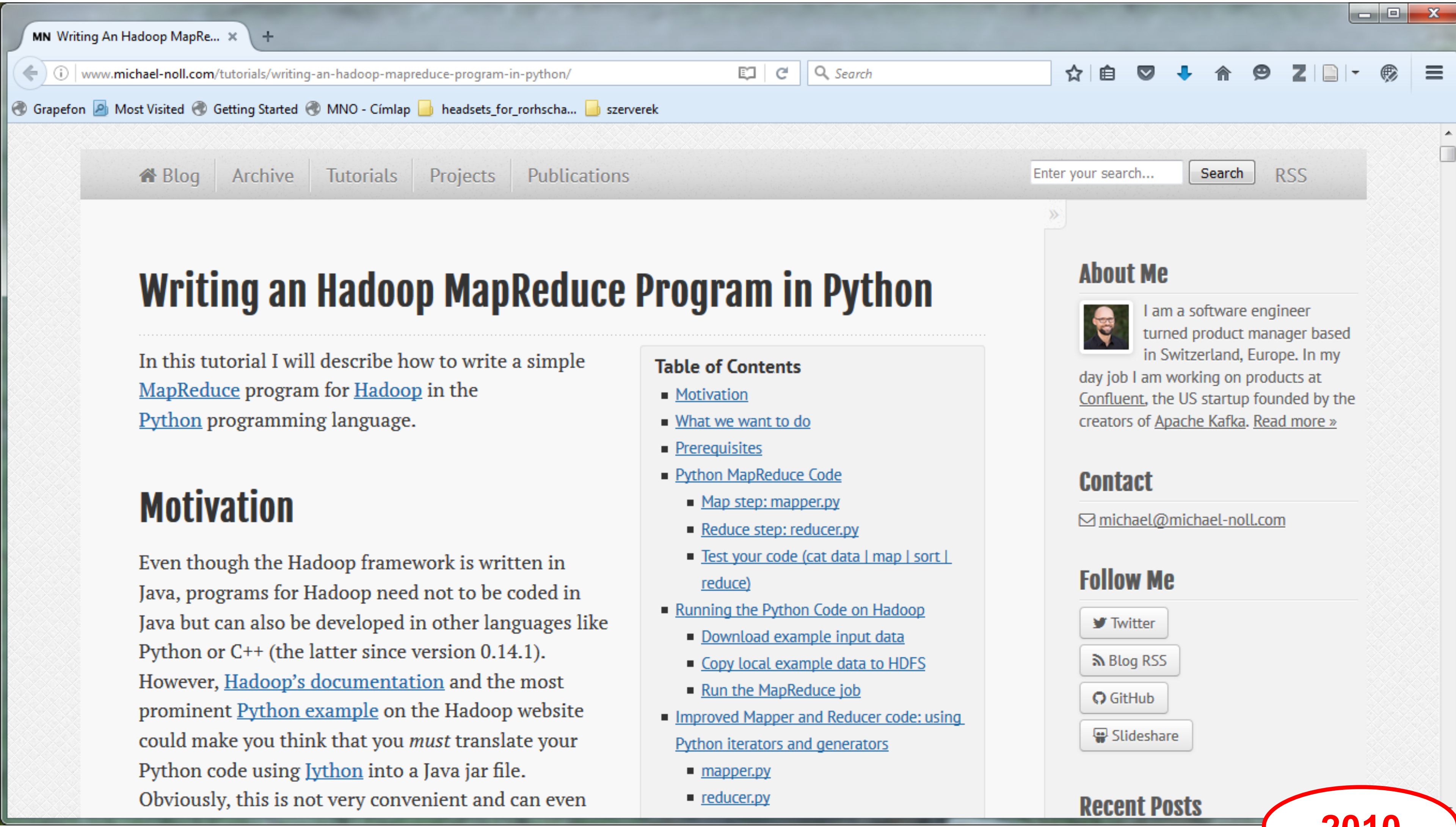
Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

In Pig Latin

```
Users    = load ‘users’ as (name, age);
Filtered = filter Users by
                  age >= 18 and age <= 25;
Pages    = load ‘pages’ as (user, url);
Joined   = join Filtered by name, Pages by user;
Grouped  = group Joined by url;
Summed   = foreach Grouped generate group,
                   count(Joined) as clicks;
Sorted   = order Summed by clicks desc;
Top5     = limit Sorted 5;

store Top5 into ‘top5sites’;
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt



Writing an Hadoop MapReduce Program in Python

In this tutorial I will describe how to write a simple MapReduce program for Hadoop in the Python programming language.

Motivation

Even though the Hadoop framework is written in Java, programs for Hadoop need not to be coded in Java but can also be developed in other languages like Python or C++ (the latter since version 0.14.1). However, Hadoop's documentation and the most prominent Python example on the Hadoop website could make you think that you *must* translate your Python code using Jython into a Java jar file. Obviously, this is not very convenient and can even

About Me

I am a software engineer turned product manager based in Switzerland, Europe. In my day job I am working on products at Confluent, the US startup founded by the creators of Apache Kafka. Read more »

Contact

michael@michael-noll.com

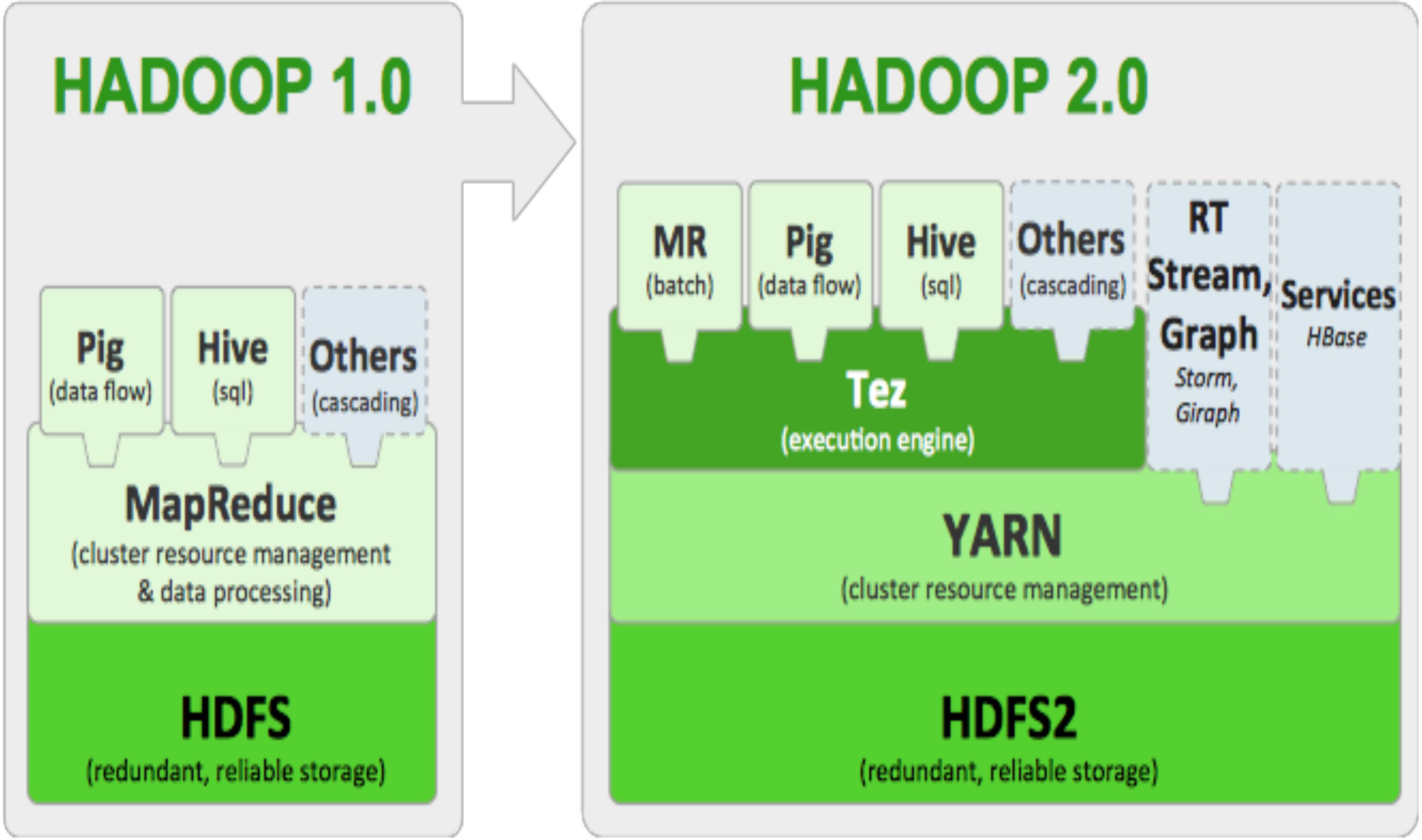
Follow Me

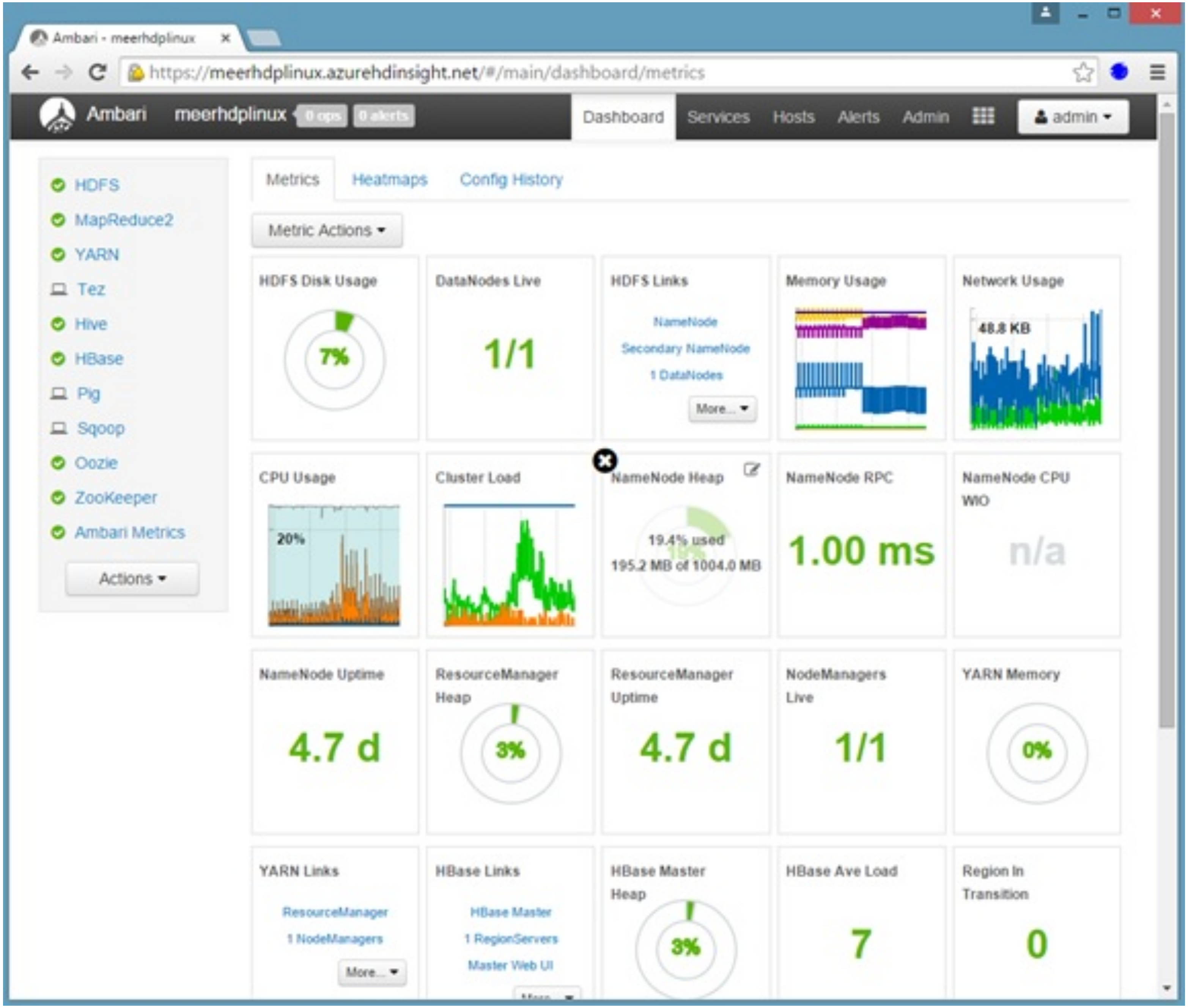
Recent Posts

2010

Limitations of MapReduce

- Linear dataflow:
  - read **input data from disk**,
  - map, reduce…,
  - **store result on disk**
- **Iterations** - when you need to process data again and again
- When your processing requires lot of data to be **shuffled** over the network.
- **Real-time** processing.
- Complex algorithms
- Processing graphs
- ….


HADOOP 1.0
Pig
(data flow)
Hive
(sql)
Others
(cascading)
MapReduce
(cluster resource management
& data processing)
HDFS
(redundant, reliable storage)
HADOOP 2.0
MR
(batch)
Pig
(data flow)
Hive
(sql)
Others
(cascading)
RT
Stream,
Graph
Storm,
Giraph
Services
HBase
Tez
(execution engine)
YARN
(cluster resource management)
HDFS2
(redundant, reliable storage)



HDFS Disk Usage: 7%

DataNodes Live: 1/1

HDFS Links: NameNode, Secondary NameNode, 1 DataNodes

Memory Usage

Network Usage: 48.8 KB

CPU Usage: 20%

Cluster Load

NameNode Heap: 19.4% used, 195.2 MB of 1004.0 MB

NameNode RPC: 1.00 ms

NameNode CPU WIO: n/a

NameNode Uptime: 4.7 d

ResourceManager Heap: 3%

ResourceManager Uptime: 4.7 d

NodeManagers Live: 1/1

YARN Memory: 0%

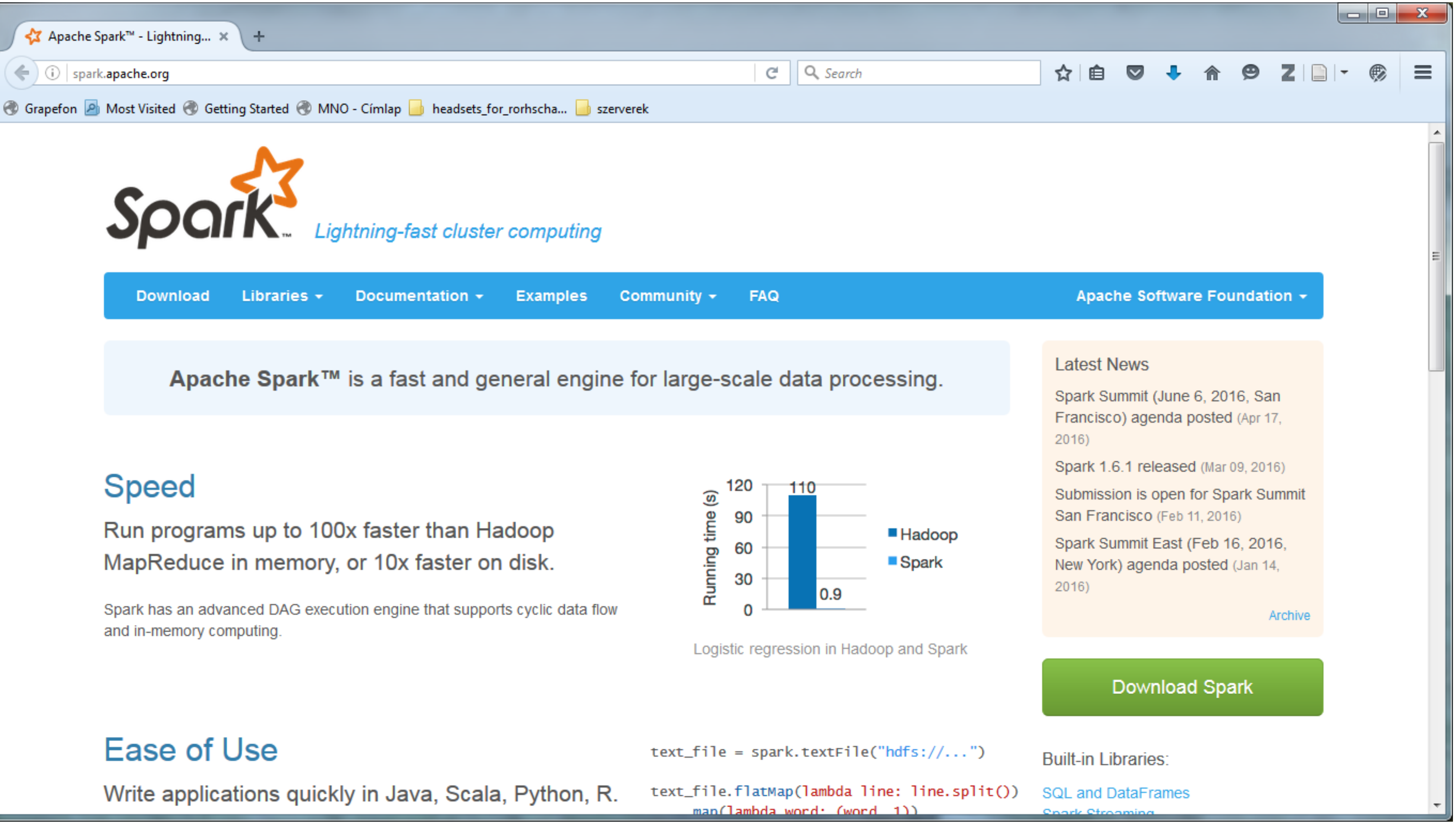
YARN Links: ResourceManager, 1 NodeManagers

HBase Links: HBase Master, 1 RegionServers, Master Web UI

HBase Master Heap: 3%

HBase Ave Load: 7

Region In Transition: 0



Spark

Lightning-fast cluster computing

Apache Spark™ is a fast and general engine for large-scale data processing.

Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.

Logistic regression in Hadoop and Spark

Ease of Use

Write applications quickly in Java, Scala, Python, R.

```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
```

Latest News

Spark Summit (June 6, 2016, San Francisco) agenda posted (Apr 17, 2016)

Spark 1.6.1 released (Mar 09, 2016)

Submission is open for Spark Summit San Francisco (Feb 11, 2016)

Spark Summit East (Feb 16, 2016, New York) agenda posted (Jan 14, 2016)

Archive

Built-in Libraries:

SQL and DataFrames

Spark Streaming

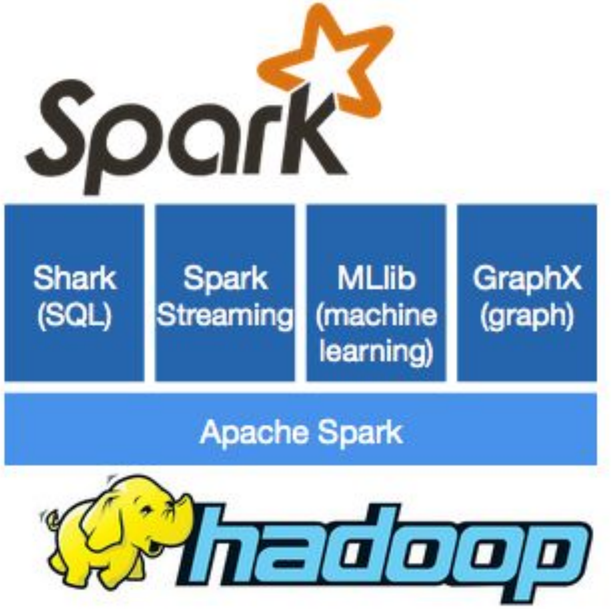
Spark

- Resilient Distributed Dataset (RDD)
  - Read-Only
  - Distributed
  - Fault-tolerant
  - Caching can be controlled

  - Iterative algorithms
  - Interactive/exploratory data analysis

Spark

- Standalone (native Spark cluster)
- Hadoop YARN
- …



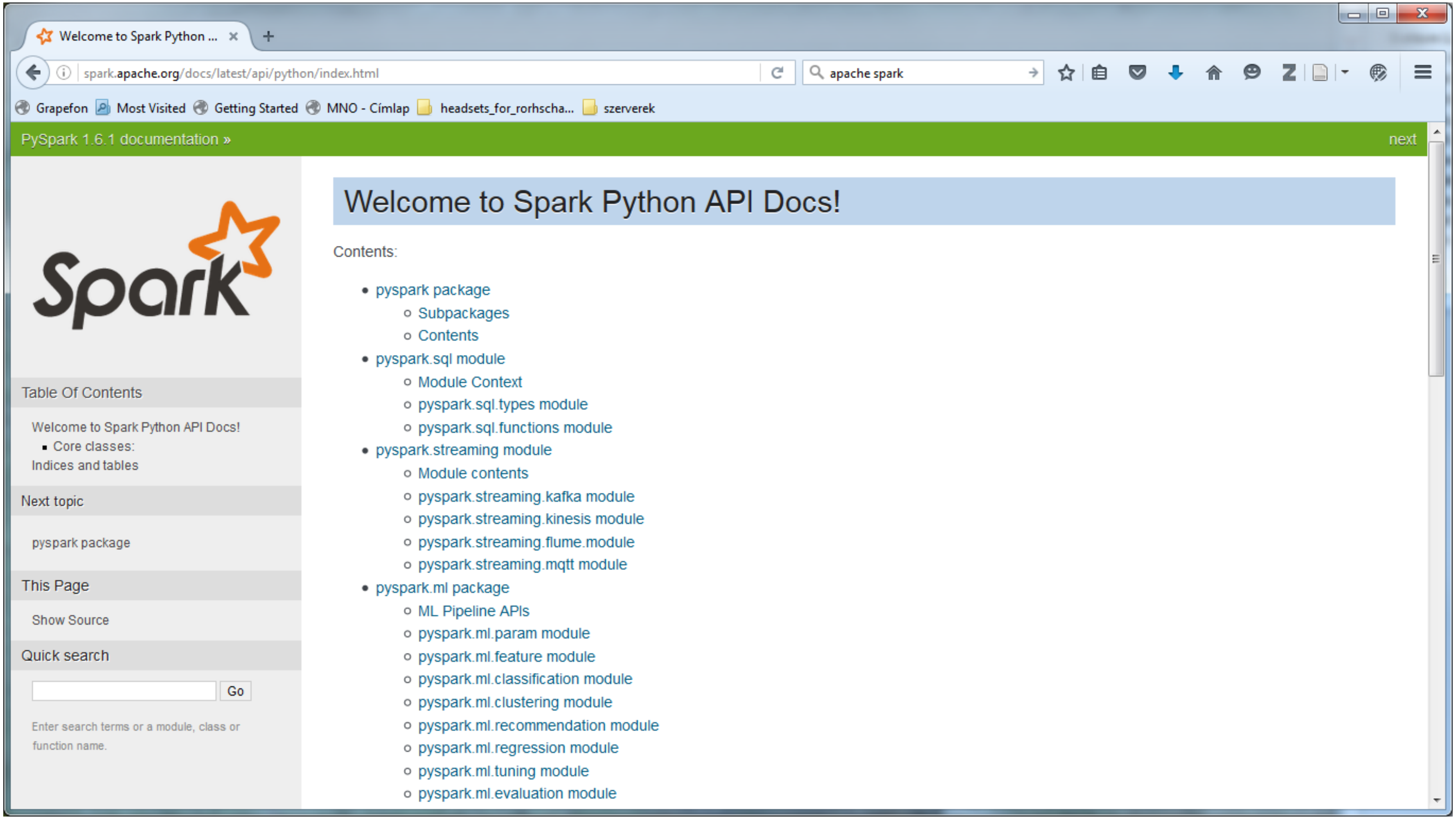
Spark disztribúciók, cloud

- Hortonworks
- Cludera

- Felhőszolgáltatók

Szolgáltatás	Google	Microsoft	Amazon
SQL adatbázis	+	+	+
NoSQL adatbázis	Hbase	Hbase, DocumentDb	Amazon DynamoDB
Disztribúciók	Hortonworks, Cloudera	Hortonworks beépítve, Cloudera	-
Gépi tanulás	-	+	+
Különböző védelmi eszközök	+	+	+

Spark - Python


Welcome to Spark Python ...
spark.apache.org/docs/latest/api/python/index.html
apache spark
Grapefon
Most Visited
Getting Started
MNO - Címlap
headsets_for_rorhscha...
szerverek
PySpark 1.6.1 documentation »
next
Spark

Welcome to Spark Python API Docs!

Contents:

- pyspark package
  - Subpackages
  - Contents
- pyspark.sql module
  - Module Context
  - pyspark.sql.types module
  - pyspark.sql.functions module
- pyspark.streaming module
  - Module contents
  - pyspark.streaming.kafka module
  - pyspark.streaming.kinesis module
  - pyspark.streaming.flume.module
  - pyspark.streaming.mqtt module
- pyspark.ml package
  - ML Pipeline APIs
  - pyspark.ml.param module
  - pyspark.ml.feature module
  - pyspark.ml.classification module
  - pyspark.ml.clustering module
  - pyspark.ml.recommendation module
  - pyspark.ml.regression module
  - pyspark.ml.tuning module
  - pyspark.ml.evaluation module

Table Of Contents

Welcome to Spark Python API Docs!
- Core classes:

Indices and tables

Next topic

pyspark package

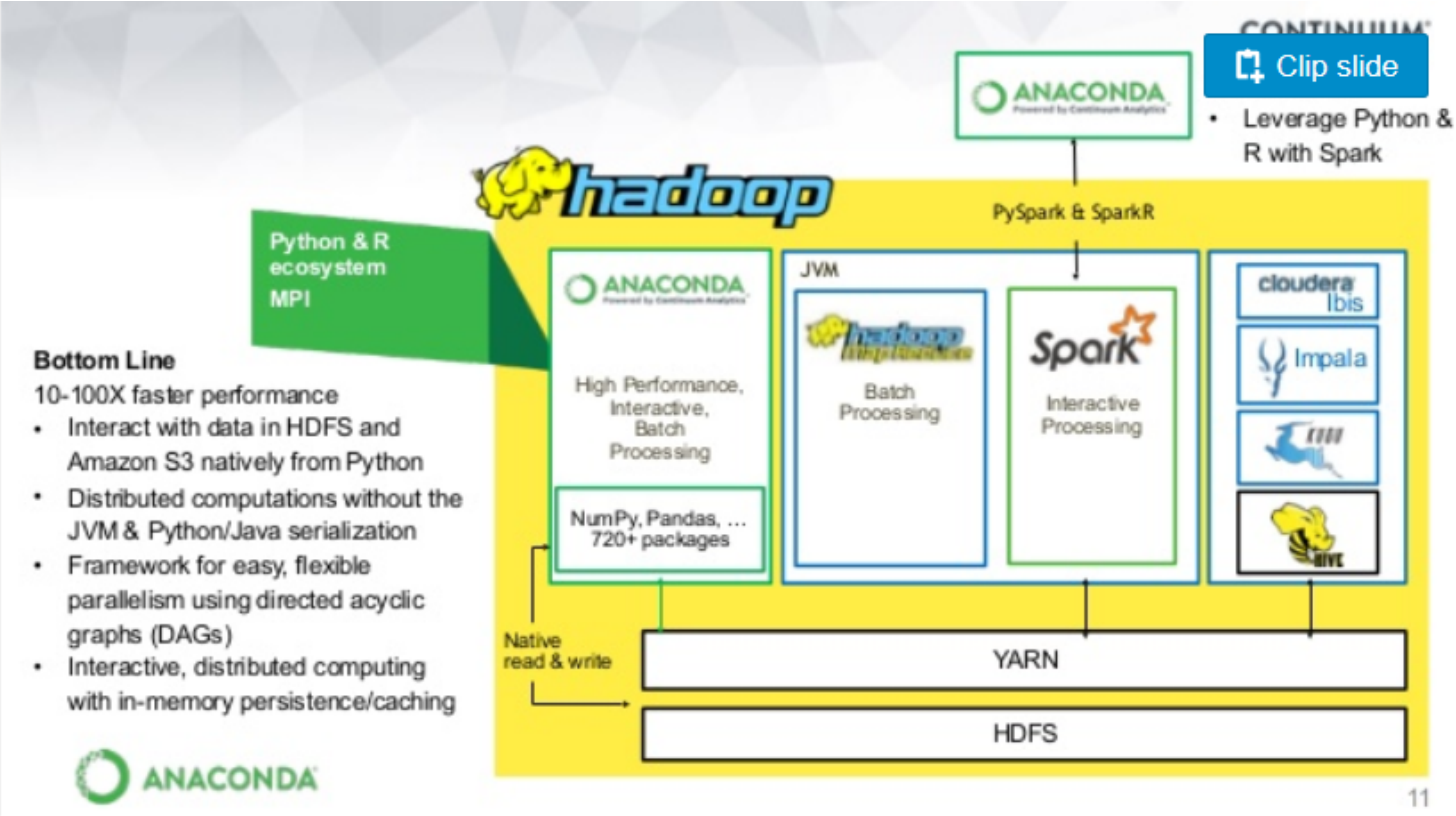
This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.



ANACONDA

- Leverage Python & R with Spark

hadoop

PySpark & SparkR

Python & R ecosystem
MPI

Bottom Line

10-100X faster performance

- Interact with data in HDFS and Amazon S3 natively from Python
- Distributed computations without the JVM & Python/Java serialization
- Framework for easy, flexible parallelism using directed acyclic graphs (DAGs)
- Interactive, distributed computing with in-memory persistence/caching

ANACONDA

High Performance, Interactive, Batch Processing

NumPy, Pandas, … 720+ packages

JVM

hadoop MapReduce

Batch Processing

Spark

Interactive Processing

cloudera Ibis

Impala

HIVE

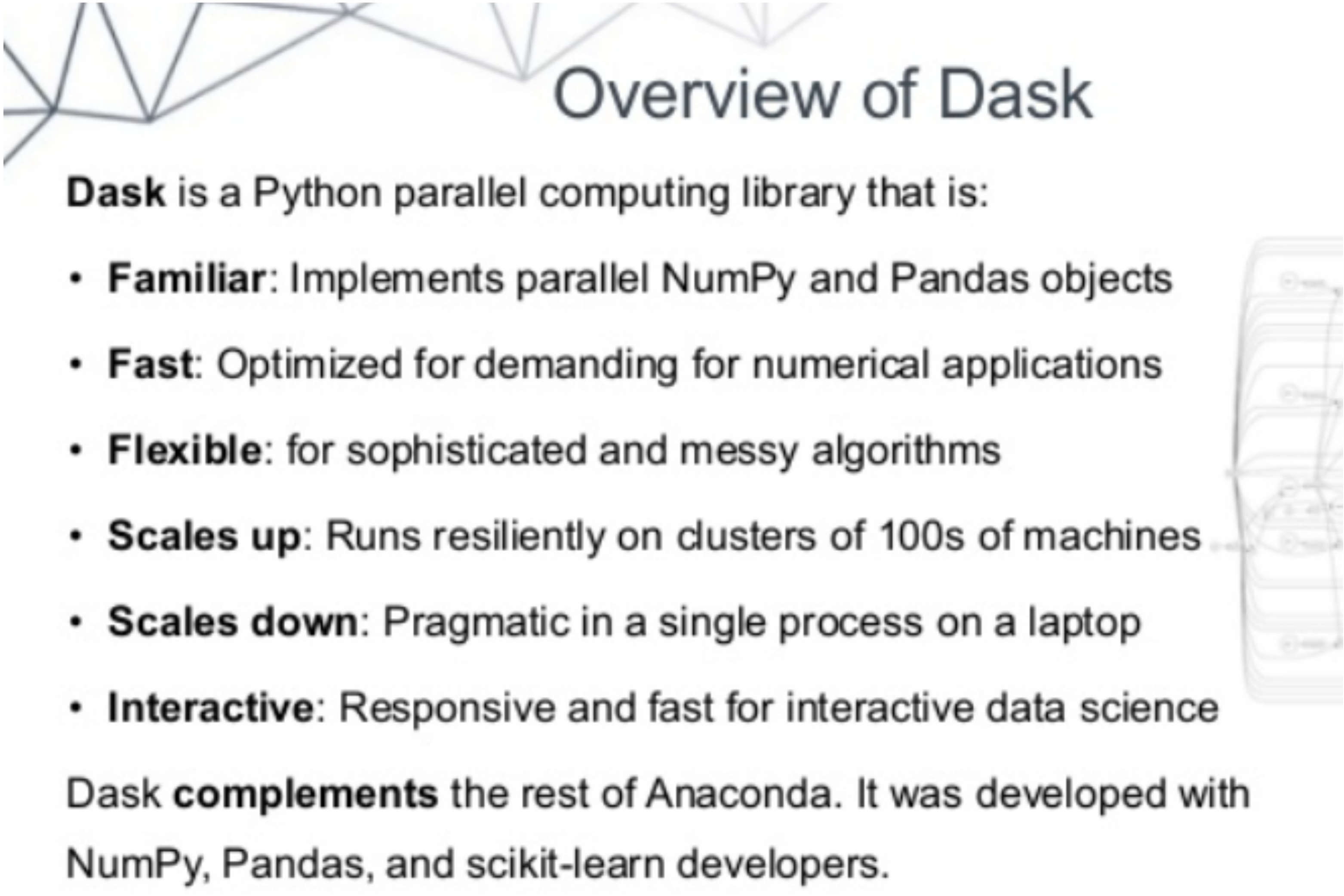
Native read & write

YARN

HDFS

ANACONDA

11

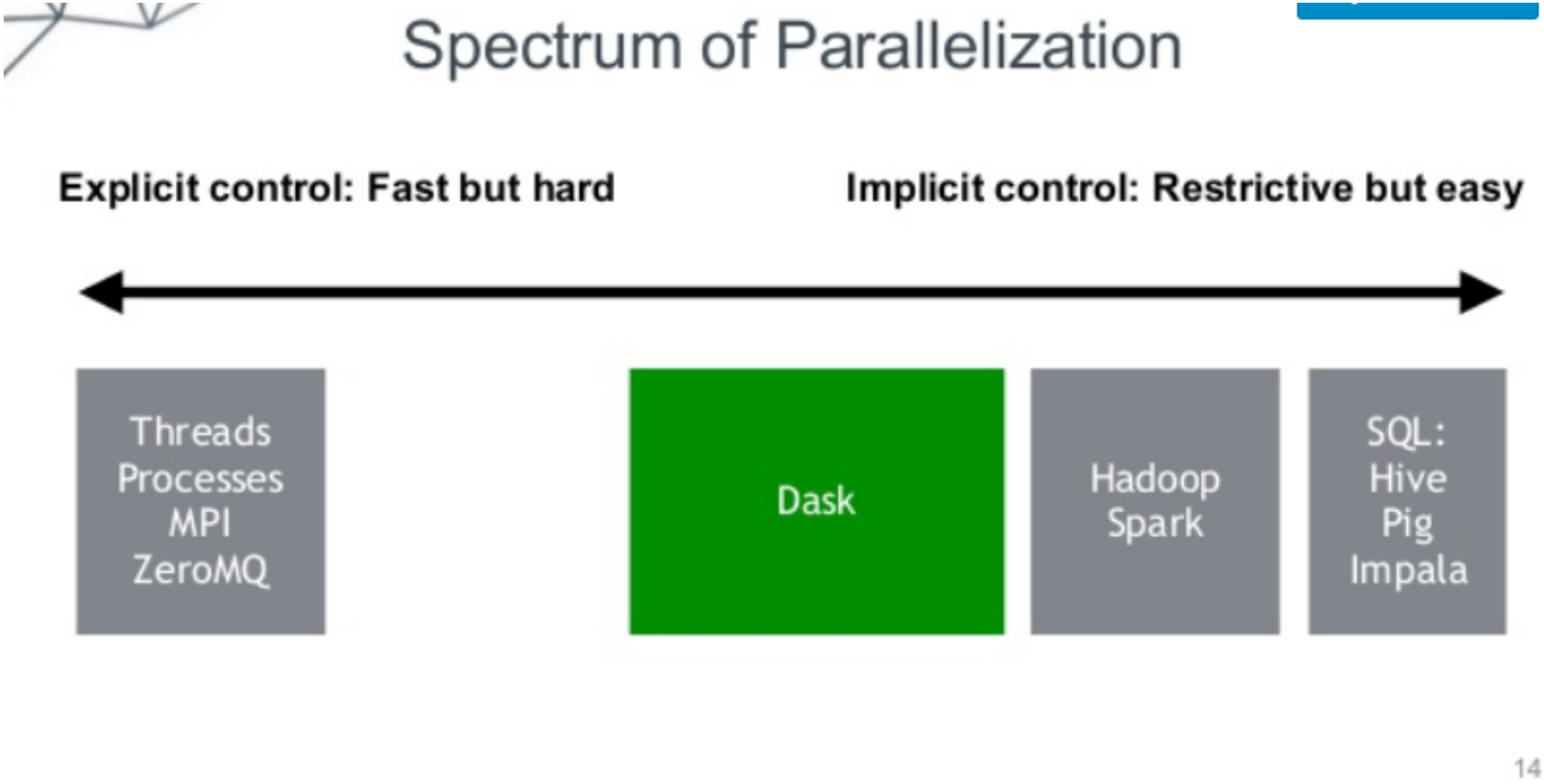


Overview of Dask

Dask is a Python parallel computing library that is:

- **Familiar**: Implements parallel NumPy and Pandas objects
- **Fast**: Optimized for demanding for numerical applications
- **Flexible**: for sophisticated and messy algorithms
- **Scales up**: Runs resiliently on clusters of 100s of machines
- **Scales down**: Pragmatic in a single process on a laptop
- **Interactive**: Responsive and fast for interactive data science

Dask **complements** the rest of Anaconda. It was developed with NumPy, Pandas, and scikit-learn developers.



Spectrum of Parallelization

Explicit control: Fast but hard

Implicit control: Restrictive but easy

Threads
Processes
MPI
ZeroMQ

Dask

Hadoop
Spark

SQL:
Hive
Pig
Impala

14