Introduction to Hadoop

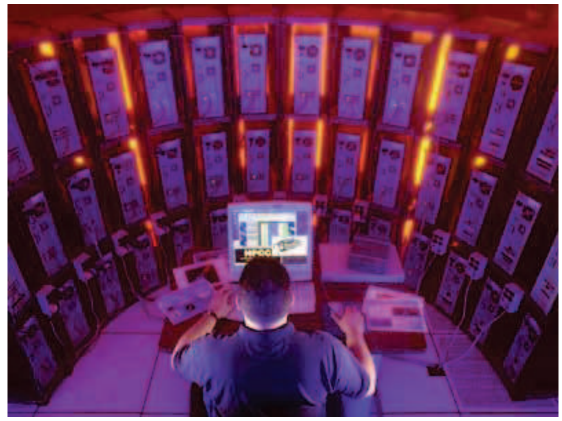
Agenda

- Introduction
  - What is distributed computing?
- What is Hadoop?
  - Comparison with RDBMS
  - Installation requirements and best practices
- Hadoop architecture
  - MapReduce
  - HDFS
  - Hadoop Common
  - Ecosystem of related projects
    - Pig, Hive, Jaql
    - Other projects

What is Distributed Computing?

- Multiple computers appear as one super computer, communicate with each other by message passing, operate together to achieve a common goal

- Challenges
  - Heterogeneity
  - Openness
  - Security
  - Scalability
  - Concurrency
  - Fault tolerance
  - Transparency



- Biggest challenge: abstract details and complexities, present users with a unified interface to manage the system

What is Hadoop?

- Apache open source software framework for reliable, scalable, distributed computing of massive amount of data
  - Hides underlying system details and complexities from user
  - Developed in Java

- Consists of 3 sub projects:
  - MapReduce
  - Hadoop Distributed File System a.k.a. HDFS
  - Hadoop Common

- Supported by several Hadoop-related projects
  - HBase
  - Zookeeper
  - Avro
  - Etc.

- Meant for heterogeneous commodity hardware

Hadoop is not for all types of work

- Not to process transactions (random access)
- Not good when work cannot be parallelized
- Not good for low latency data access
- Not good for processing lots of small files
- Not good for intensive calculations with little data

PureData Ecosystem

Who uses Hadoop?

RDBMS vs Hadoop

	RDBMS	Hadoop
Data sources	Structured data with known schemas	Unstructured and structured
Data type	Records, long fields, objects, XML	Files
Data Updates	Updates allowed	Only inserts and deletes
Language	SQL & XQuery	Pig (Pig Latin), Hive (HiveQL), Jaql
Processing type	Quick response, random access	Batch processing
Data integrity	Data loss is not acceptable	Data loss can happen sometimes
Security	Security and auditing	Partial
Compress	Sophisticated data compression	Simple file compression
Hardware	Enterprise hardware	Commodity hardware
Data access	Random access (indexing)	Access files only (streaming)
History	~40 years of innovation	< 5 years old
Community	Widely used, abundant resources	Not widely adopted yet

Warehouse vs Hadoop

	Data Warehouse	Hadoop
Data sources	Structured, high value data. Pre - Processed	Unstructured and structured
Data type	Records, long fields, objects, XML	Files
Data Updates	Updates allowed	Only inserts and deletes
Language	Vendor specific	Pig (Pig Latin), Hive (HiveQL), Jaql
Processing type	Batch Processing	Batch processing
Data integrity	Data loss is not acceptable	Data loss can happen sometimes
Security	Security and auditing	Partial
Compress	Sophisticated data compression	Simple file compression
Hardware	Enterprise hardware	Commodity hardware
Data access	Random access (indexing)	Access files only (streaming)
History	~20 years of innovation	< 5 years old
Community	Widely used, abundant resources	Not widely adopted yet

Hadoop – Installation Requirements

- Installation types:
  - Single-node:
    - simple operations
    - local testing and debugging
  - Multi-node cluster:
    - production level operation
    - thousands of nodes
- Hardware:
  - Can use commodity hardware
  - Best practice:
    - **RAM**: MapReduce jobs mostly I/O bound, plan enough RAM
    - **CPU**: high-end CPUs are often not cost-effective
    - **Disks**: use high capacity disks as Hadoop is storage hungry
    - **Network**: depends on workload, consider high-end network gear for large clusters
- Software**:**
  - OS:
    - GNU / Linux for development and production
    - Windows / Mac for development
  - Java
  - ssh

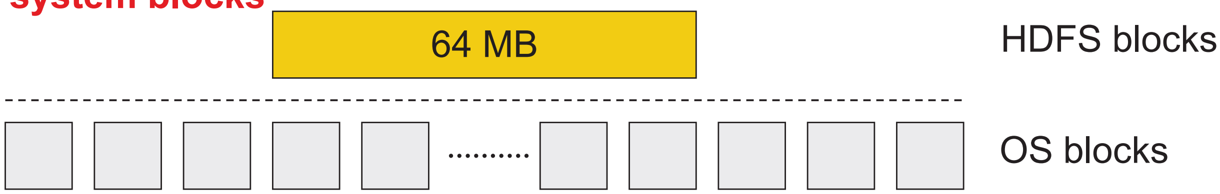
PureData Ecosystem

Hadoop Distributed File System (HDFS)

- Distributed, scalable, fault tolerant, high throughput
- Data access through MapReduce
- Files split into blocks
- 3 replicas for each piece of data by default
- Can create, delete, copy, but NOT update
- Designed for streaming reads, not random access
- Data locality: processing data on or near the physical storage to decrease transmission of data

HDFS – Blocks

- **HDFS is designed to support very large files**
- **Each file is split into blocks**
  – Hadoop default: 64MB
  – BigInsights default: 128MB
- **Blocks reside on different physical DataNode**
- **Behind the scenes, 1 HDFS block is supported by multiple operating system blocks**



- **If a file or a chunk of the file is smaller than the block size, only needed space is used. E.g.: a 210MB file is split as**

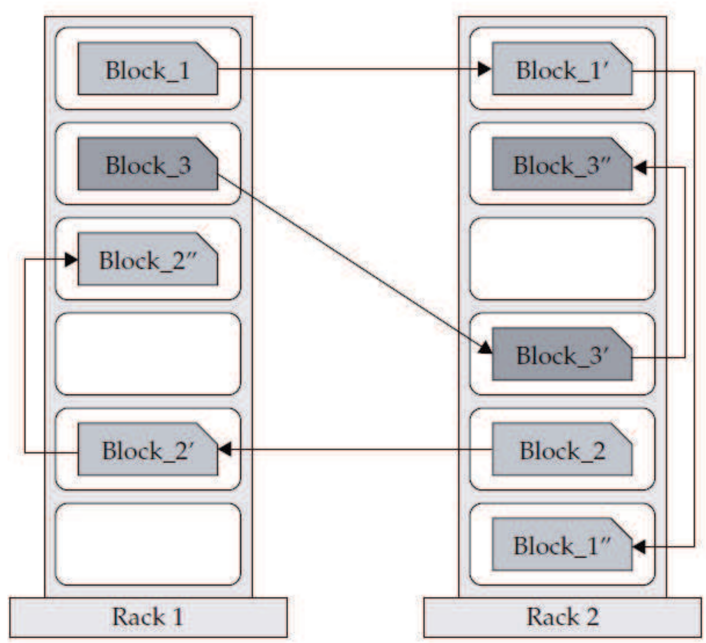
HDFS – Replication

- **Blocks of data are replicated to multiple nodes**
  – Behavior is controlled by replication factor, configurable per file
  – Default is 3 replicas

Common case:

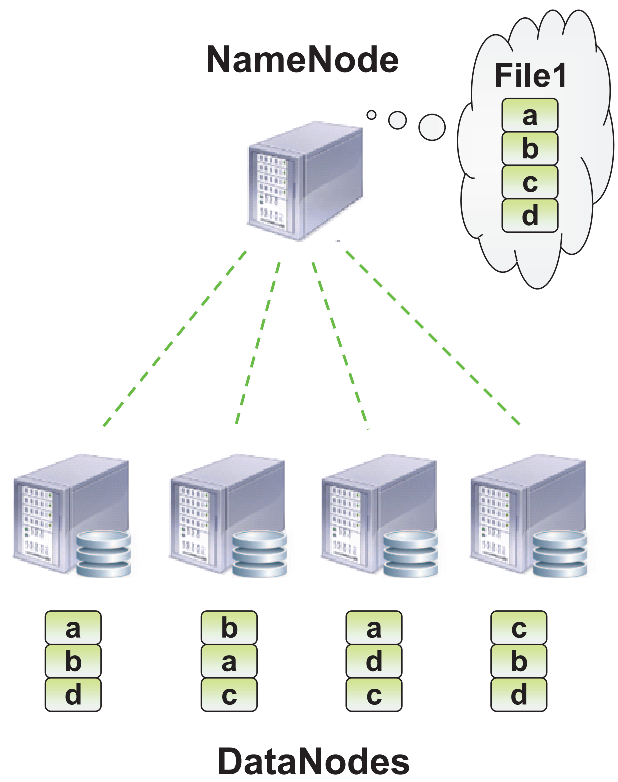
- **one replica on one node in the local rack**
- **another on a node in a different (remote) rack**
- **and the last on a different node in the same remote rack**

This cuts inter-rack network bandwidth, which improves write performance



HDFS – Architecture

- **Master / Slave architecture**
- **Master: NameNode**
  - manages the file system namespace and metadata
    - FsImage
    - EditLog
  - regulates access by files by clients
- **Slave: DataNode**
  - many per cluster
  - manages storage attached to the nodes
  - periodically reports status to NameNode



MapReduce

- Distributed computing framework that takes advantage of data locality to push the computation to the data
  - Distributed computing: clusters of computers with local memory and disk
    - Network intensive for big data
  - Parallel computing: multiple CPUs processing over shared memory and file system
- By decomposing the tasks we can achieve parallelism…
  - Map: works independently to convert input data into key-pair values.

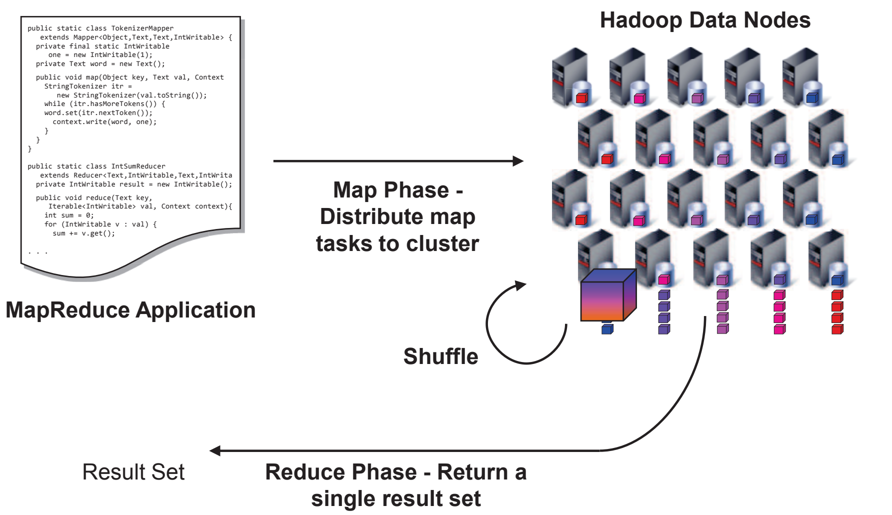
(k1,v1) => list(k2,v2)

  - Reduce: works independently on all values for a give key and transforms them to a single output set per key

(k2, list(v2) => list(v3)

MapReduce Explained

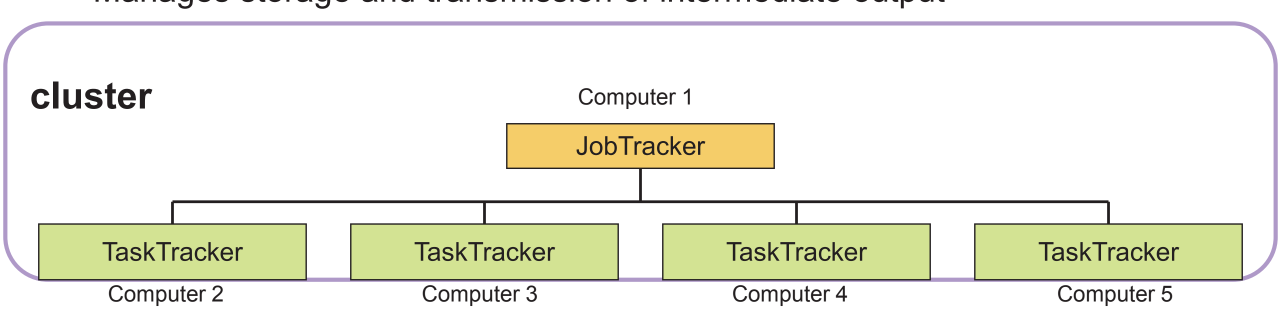
- Hadoop computation model
  - Data stored in a distributed file system spanning many inexpensive computers
  - Bring function to the data
  - Distribute application to the compute resources where the data is stored
- Scalable to thousands of nodes and petabytes of data



1. Map Phase
   (break job into small parts)
2. Shuffle
   (transfer interim output for final processing)
3. Reduce Phase
   (boil all output down to a single result set)

MapReduce Engine

- Master / Slave architecture
  - Single master (JobTracker) controls job execution on multiple slaves (TaskTrackers).
- **JobTracker**
  - Accepts MapReduce jobs submitted by clients
  - Pushes *map* and *reduce* tasks out to TaskTracker nodes
  - Keeps the work as physically close to data as possible
  - Monitors tasks and TaskTracker status
- **TaskTracker**
  - Runs map and reduce tasks
  - Reports status to JobTracker
  - Manages storage and transmission of intermediate output

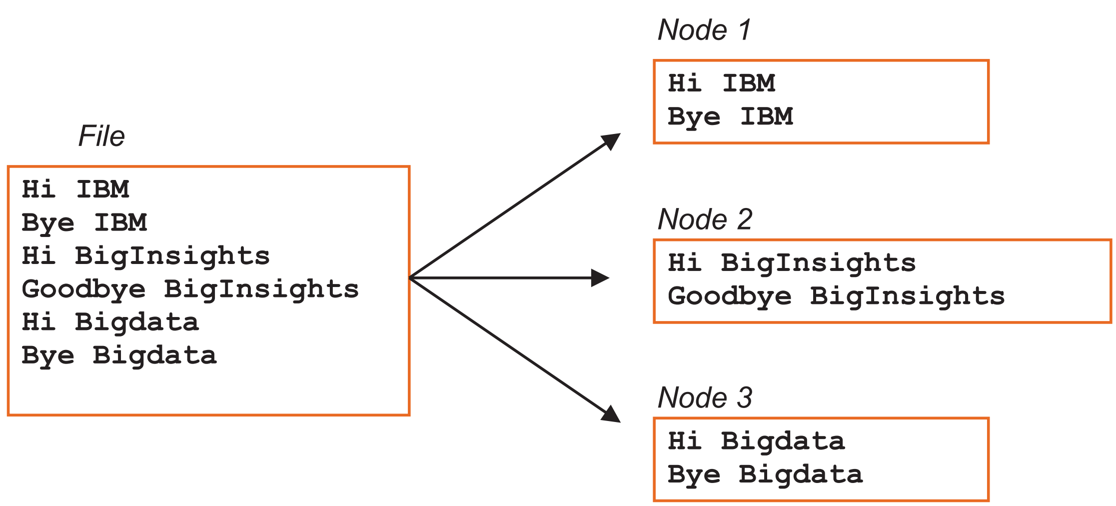


MapReduce Examples

- Map
  - **Word Count**
    - Read the text from a stream of text (i.e.: files) and emit each word as a key with value 1.
  - **Inverted Index**
    - Read the text from a stream of documents and emit each word as a key in the document.
  - **Maximum Temperature**
    - Read formatted data and emit year as a key with temperature as value.
  - **Mean Rain Precipitation**
    - Read daily data and emit (year/month, lat, long) as key with temperature as value.
- Reduce
  - **Operations such as count, list, max, average, etc.**
    - Set values for each key

MapReduce

- **Example: word count**
  - The map function emits each word plus an associated count of occurrences, 1 in this example



MapReduce

- **Example: word count**
  - Map step
    - The map function emits each word plus an associated count of occurrences, 1 in this example

Node 1

```
Hi IBM          →   <Hi, 1>
Bye IBM             <IBM, 1>
                    <Bye, 1>
                    <IBM, 1>
```

Node 2

```
Hi BigInsights          →   <Hi, 1>
Goodbye BigInsights         <BigInsights, 1>
                            <Goodbye, 1>
                            <BigInsights, 1>
```

Node 3

```
Hi Bigdata      →   <Hi, 1>
Bye Bigdata         <Bigdata, 1>
                    <Bye, 1>
                    <Bigdata, 1>
```


MapReduce

- **Example: word count (continued)**
  - Shuffle
    - Locally sort the intermediary output tuples by key and aggregate values

```
<Hi, 1>             →   <Bye, 1>
<IBM, 1>                <Hi, 1>
<Bye, 1>                <IBM, [1, 1]>
<IBM, 1>

<Hi, 1>             →   <Goodbye, 1>
<BigInsights, 1>        <Hi, 1>
<Goodbye, 1>            <BigInsights, [1, 1]>
<BigInsights, 1>

<Hi, 1>             →   <Bye, 1>
<Bigdata, 1>            <Bigdata, [1, 1]>
<Bye, 1>                <Hi, 1>
<Bigdata, 1>
```

MapReduce

- **Example: word count (continued)**
  - Reduce
    - Sums up values

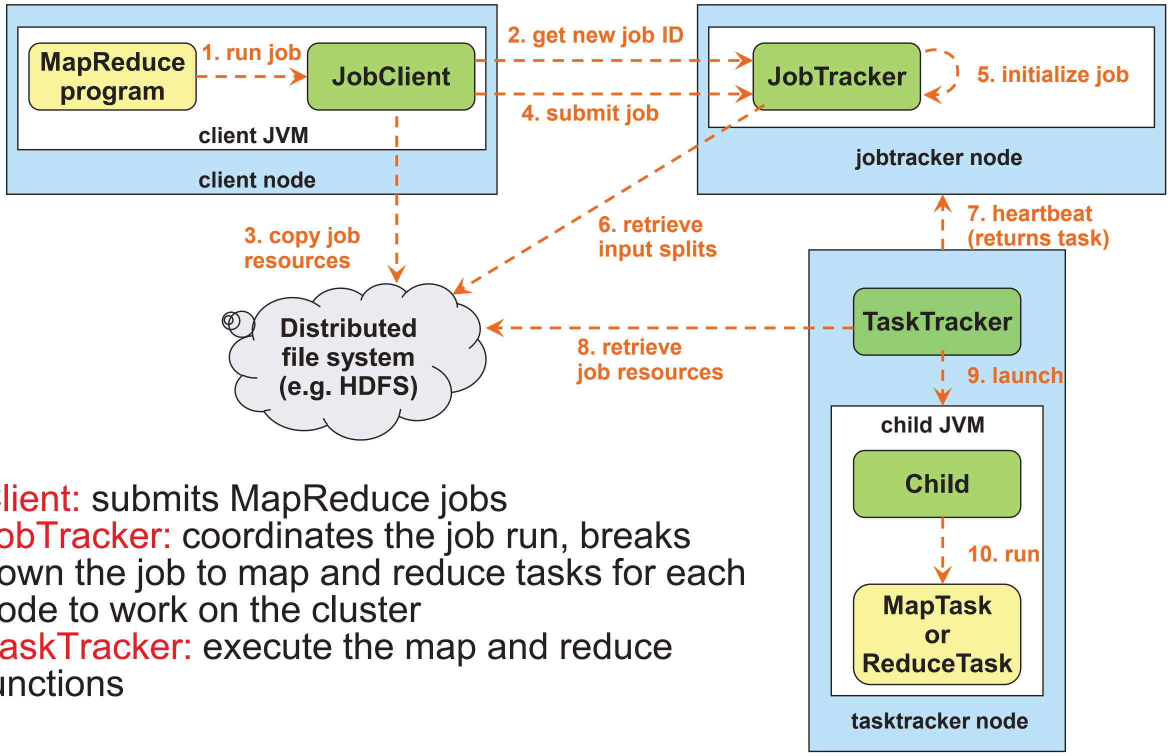
```
<Bye, 1>
<Hi, 1>
<IBM, [1, 1]>

<Goodbye, 1>
<Hi, 1>
<BigInsights, [1, 1]>

<Bye, 1>
<Bigdata, [1, 1]>
<Hi, 1>
```

```
<Bye, 2>
<Goodbye, 1>
<IBM, 2>
<Hi, 3>
<BigInsights, 2>
<Bigdata, 2>
```


MapReduce Fault Tolerance

- If a task dies
  - Retry on another node
    - Map not affected because it has no dependencies.
    - Reduce not affected because map outputs are stored on disk.
- If a node dies
  - Restart the current tasks on another node.
  - Re-execute the maps the node previously ran.

How does Hadoop run MapReduce jobs?



- Client: submits MapReduce jobs
- JobTracker: coordinates the job run, breaks down the job to map and reduce tasks for each node to work on the cluster
- TaskTracker: execute the map and reduce functions

Hadoop Common

- Formerly known as Hadoop Core
- Contains common utilities and libraries that support the other Hadoop sub projects
  - File system
  - Remote Procedure Call (RPC)
  - Serialization
- E.g. file system shell
  - To interact directly with HDFS files, you need to use
    hadoop fs -<args>

```
hadoop fs –ls
```

Hadoop Open Source Projects

- Hadoop is supplemented by an ecosystem of open source projects

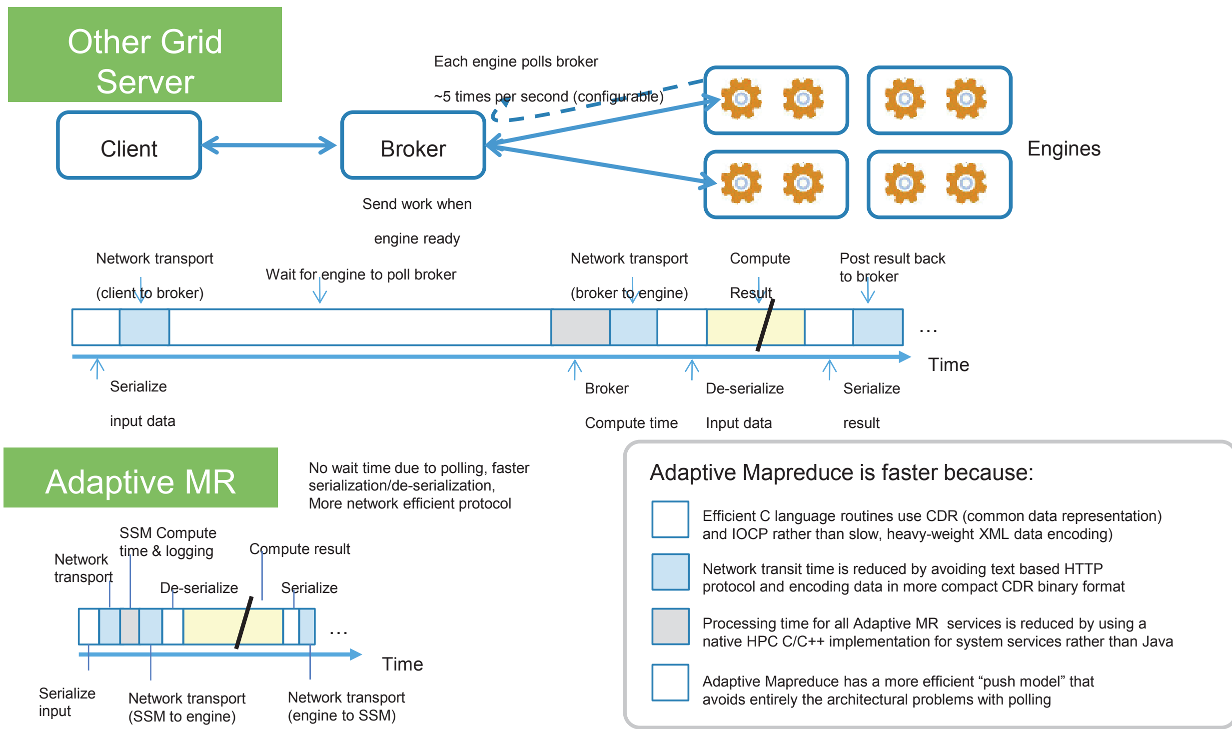
How to Analyze Large Data Sets in Hadoop

- Although the Hadoop framework is implemented in Java, MapReduce applications do not need to be written in Java

- To abstract complexities of Hadoop programming model, a few application development languages have emerged that build on top of Hadoop:
  – Pig
  – Hive
  – Jaql

Adaptive MapReduce Advantages

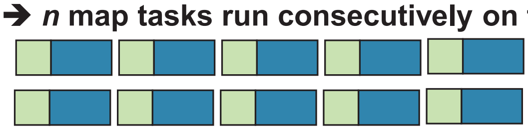
Low-latency task scheduling, optimized middleware



Design Goals of Adaptive MapReduce

- Balance workload across Map tasks
- Minimize startup and scheduling costs
  - Relatively high when operating on small files or splits
- Promote greater local aggregation
- Allow Map tasks to take on additional work until it doesn't make sense anymore

Traditional MR ➔ *n* map tasks run consecutively on the same node/slot



AdaptiveMR ➔ *One* map task might process the several splits

Legend:

Startup/Scheduling cost (e.g. loading reference data)

map task processing

Adaptive MapReduce Enhancements

- Speeds up a class of jobs (e.g., jobs that process small files)
- Accomplished by changing how certain MapReduce tasks executed
  - Mappers can decide at runtime to take on more work (until it doesn't make sense anymore)
  - Communication via ZooKeeper
- Supported on Jaql/Java jobs (not supported on Hive or Pig jobs)
- Off by default
  - Enable on Jaql jobs with a Jaql option, or
  - MapReduce job property setting

Flexible (Intelligent) Job Scheduler

- Optimize response time for small jobs
  - Available in addition to FAIR, FIFO scheduling
  - Example: What if J2 is small and J1 is huge under FIFO?