Big Data Analytics with Cloud

where *speed* matters

Dr. Posco Tso

Senior Lecturer
Department of Computer Science

Agenda

- Introduction

- MapReduce

- Big Data Infrastructure and Analytic Tools

- Cloud and Big Data Programming - Hands on

- Conclusion

LIVERPOOL
JOHN MOORES
UNIVERSITY

About Me

- PhD, City University of Hong Kong (QS 57th worldwide, 2015)

  - ✓ 1 US Patent and 1 Start-up

- SICSA Next Generation Internet Fellow, Glasgow University

  - ✓ Built a cloud and big data testbed

- Senior Lecture, Liverpool John Moores University

  - ✓ Building a low power cloud for Internet of Things

Big Data Infrastructure

- How Big is "Big Data"?

  - \> 1 TB

  - Simple C/C++ code with legacy database will beat "Big Data Analytics" systems in speed.

  - Not able to leverage parallelism

- Components for Big Data infrastructure

  - Compute cluster(s); Data analytics tools; File systems/ databases

Big Data Infrastructure

Compute Clusters	Data Analytics Tools	File Systems/ Databases
(Virtual) Machine cluster(s); High performance computing (HPC) clusters;	Hadoop (framework); Spark; Storm; MapR; Pig; …	HDFS; S3; GFS; Cassandra; HBase; BigTable; MongoDB; …

Big Data Infrastructure

Compute Clusters	Data Analytics Tools	File Systems/ Databases
(Virtual) Machine cluster(s); High performance computing (HPC) clusters;	Hadoop (framework); Spark; Storm; MapR; Pig; …	HDFS; S3; GFS; Cassandra; HBase; BigTable; MongoDB; …

Agenda

- Introduction

- **MapReduce**

- Big Data Infrastructure and Analytic Tools

- Cloud and Big Data Programming - Hands on

- Conclusion

MapReduce

- Simple data-parallel programming model designed for scalability and fault-tolerance

- Pioneered by Google for processing > 20 petabytes of data per day

- Popularised by open-source Hadoop project

"Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages."

Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

Lisp - List Processing

- Functional Programming
  - is when (mathematical) functions are used as the fundamental building blocks of a program.
- Lists are primitive data types:
  - `(1 2 3 4 5)
  - `((a 1) (b 2) (c 3))
- Functions written in prefix notation:
  - (+ 1 2) —> 3
  - (* 3 4) —> 12
  - (sqrt (+ (* 3 3) (* 4 4))) —> 5
  - (* x 5) —> 15

Example program

```
(define (factorial n))
   (if (= n 1)
      1
      (* n (factorial (- n 1)))))

(factorial 6) —> 720
```
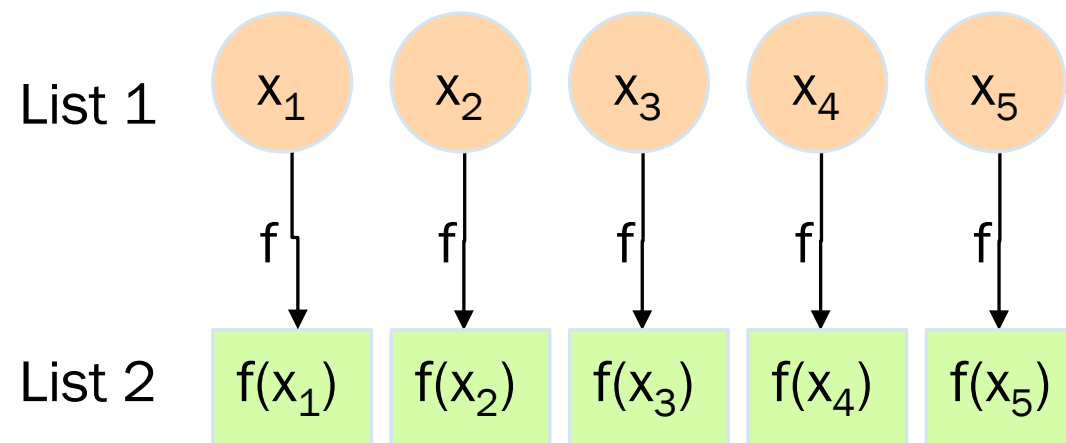
from Lisp to MapReduce

- What do functional programming and Lisp have to do with MapReduce

- There are two important concepts in functional programming

    - Map: do something to everything on a list

    - Fold: combine or accumulate results in a list

Map

- Map is a higher-order function

- How map works:
  - Function is applied to every element in a list
  - Result is a new list

List 1 $\quad$ x_1 $\quad$ x_2 $\quad$ x_3 $\quad$ x_4 $\quad$ x_5

f $\quad$ f $\quad$ f $\quad$ f $\quad$ f

List 2 $\quad$ $f(x_1)$ $\quad$ $f(x_2)$ $\quad$ $f(x_3)$ $\quad$ $f(x_4)$ $\quad$ $f(x_5)$

Fold/Reduce

- Fold is also a higher-order function

- How fold works:
    - Accumulator set to initial value
    - Function applied to list element and the accumulator
    - Result stored in the accumulator
    - Repeated for every item in the list
    - Result is the final value in the accumulator
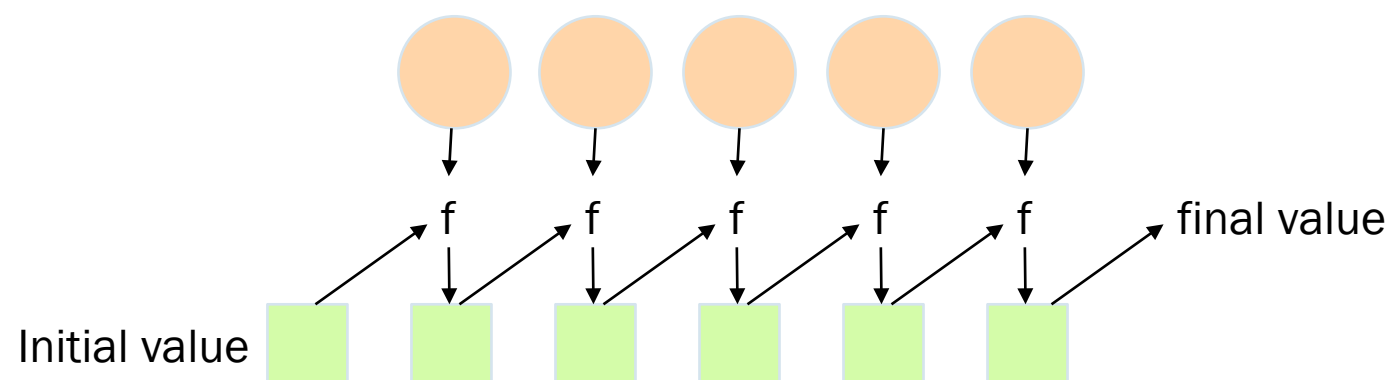
Map/Fold in Action

- Simple map example:

(map (lambda (x) (* x x))
`(1 2 3 4 5))
—> `(1 4 9 16 25)

- Fold examples:

fold + 0 `(1 2 3 4 5)) —> 15
fold * 1 `(1 2 3 4 5)) —> 120

- Sum of squares:

(define (sum-of-squares v)
(fold + 0 (map (lambda (x) (* x x)) v)))
(sum-of-squares `(1 2 3 4 5)) —> 55

from Lisp to MapReduce

- Let's assume a long list of records: imagine if …
  - We can parallelise map operations
  - We have a mechanism for bringing map results back together in the fold operation

- That's MapReduce!

- Observations:
  - No limit to map parallelisation since maps are independent
  - We can reorder folding if the fold function is commutative and associative

MapReduce Programming Model

- Data type: key-value records

- Map function:

$$(K_{in}, V_{in}) \longrightarrow \text{list } (K_{inter}, V_{inter})$$
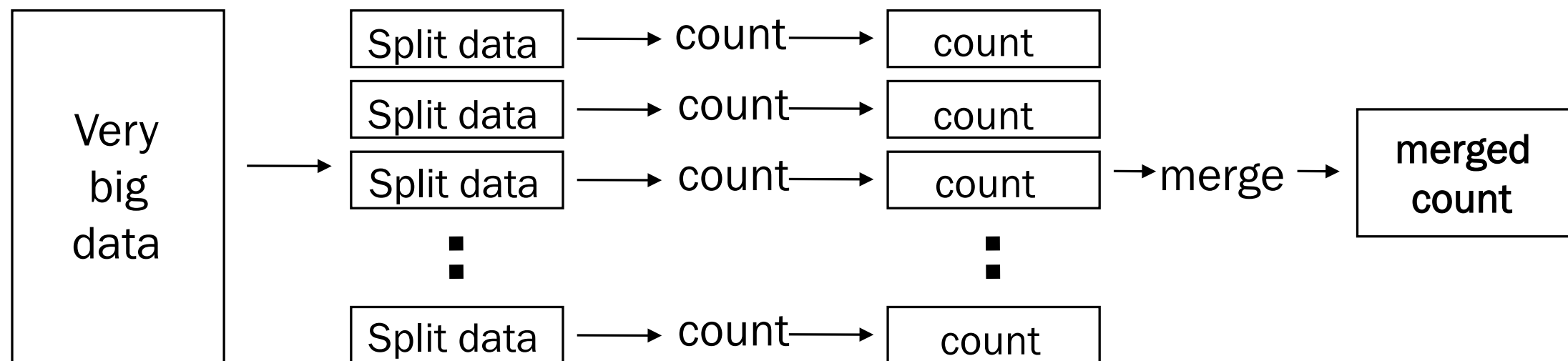
- Reduce function:

$$(K_{inter}, \text{list}(V_{inter})) \longrightarrow \text{list } (K_{out}, V_{out})$$
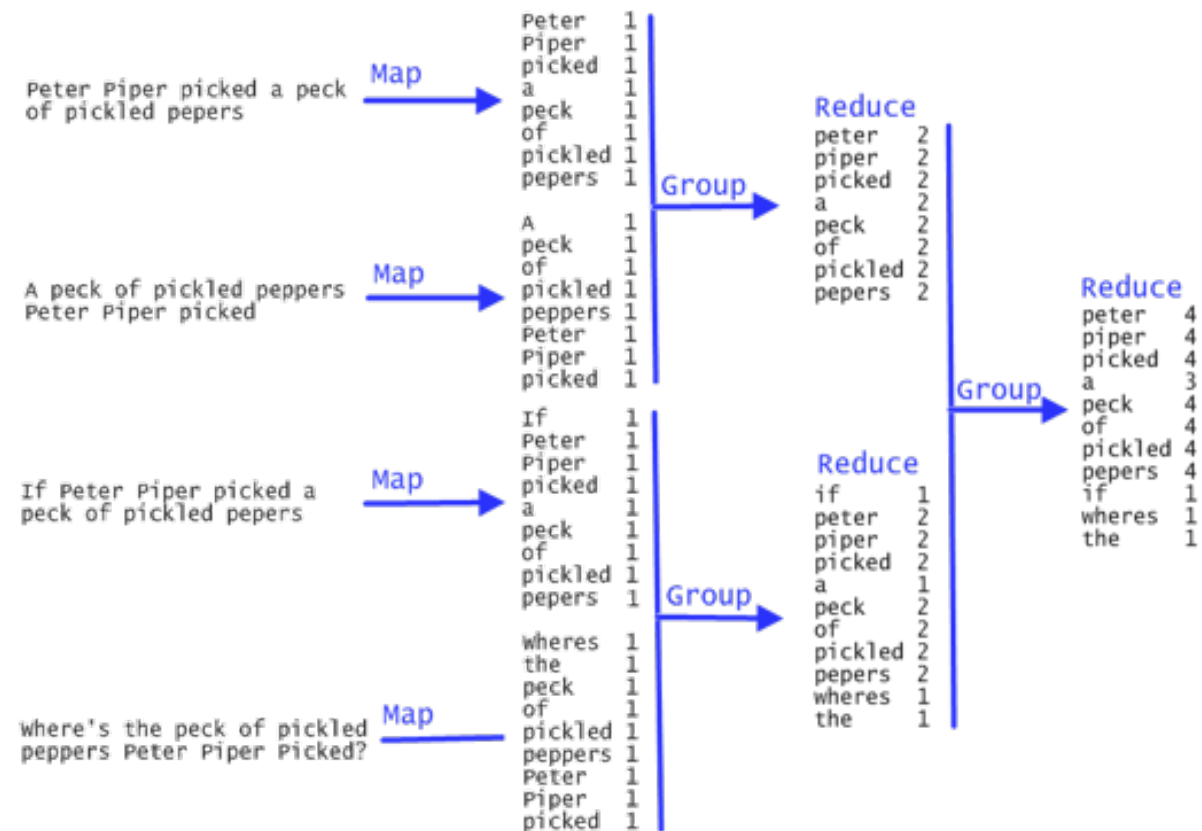
Example: Word Count

```
def mapper(line):
    foreach word in line.split():
        output(word, 1)

def reducer(key, values):
    output(key, sum(values))
```

Distributed Word Count

Distributed Word Count

Agenda

- Introduction

- MapReduce

- **Big Data Infrastructure and Analytic Tools**

- Cloud and Big Data Programming - Hands on

- Conclusion

Big Data Infrastructure

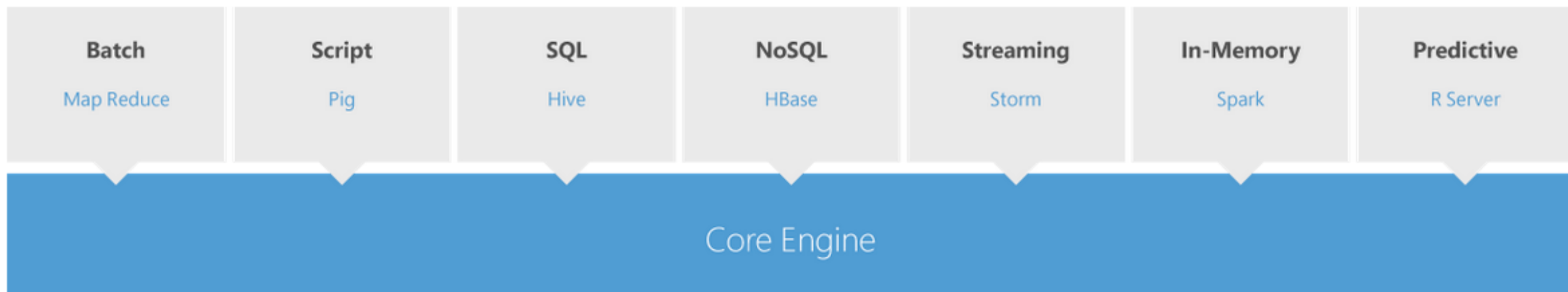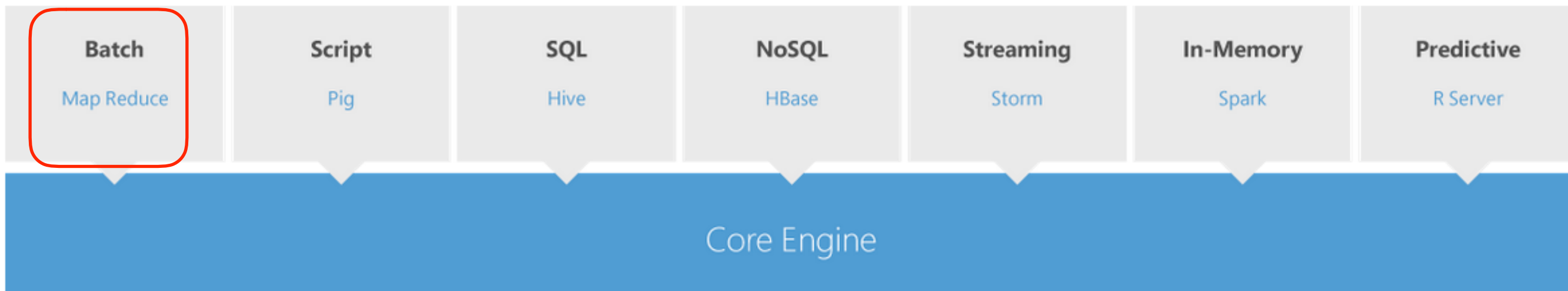| Compute Clusters	Data Analytics Tools	File Systems/ Databases
(Virtual) Machine cluster(s); High performance computing (HPC) clusters;	Hadoop (framework); Spark; Storm; MapR; Pig; …	HDFS; S3; GFS; Cassandra; HBase; BigTable; MongoDB; …

Big Data Infrastructure

Cloud Computing provides cheap and on-demand access to compute, storage and tools

Compute Clusters	Data Analytics Tools	File Systems/ Database
(Virtual) Machine cluster(s); High performance	Hadoop (framework); Spark; Storm; MapR; Pig;	HDFS; S3; GFS; Cassandra; HBase; BigTable;

Data analytic tools

Batch	Script	SQL	NoSQL	Streaming	In-Memory	Predictive
Map Reduce	Pig	Hive	HBase	Storm	Spark	R Server

Core Engine

Data analytic tools

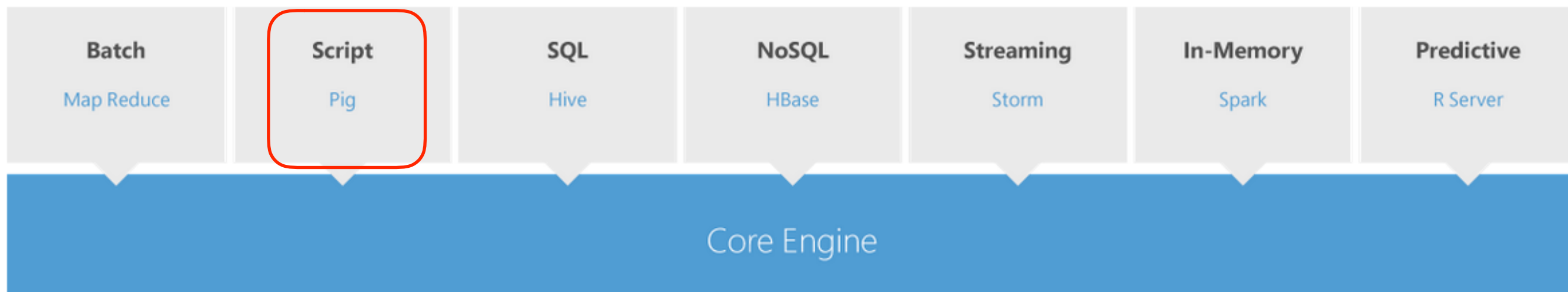| Batch	Script	SQL	NoSQL	Streaming	In-Memory	Predictive
Map Reduce	Pig	Hive	HBase	Storm	Spark	R Server

Core Engine

- Hadoop MapReduce

  - http://hadoop.apache.org/

  - Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

  - The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

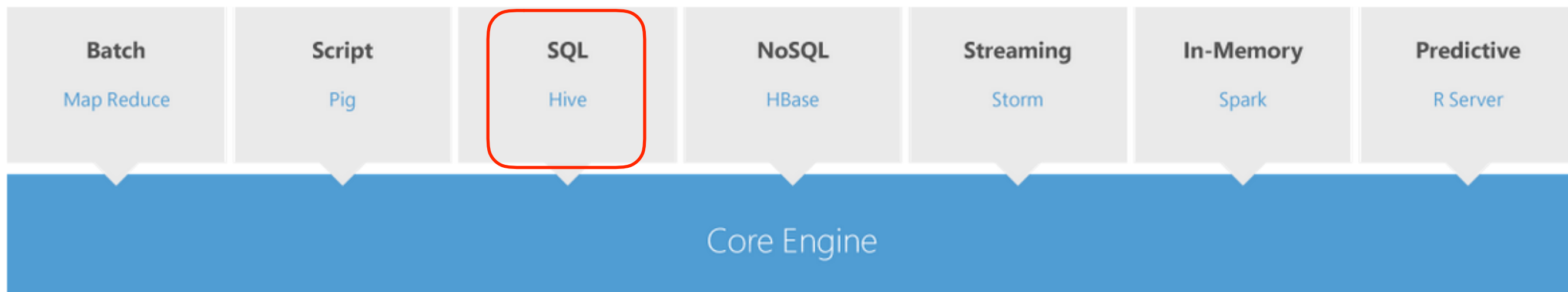 - Low level programming practice

Data analytic tools

Batch	Script	SQL	NoSQL	Streaming	In-Memory	Predictive
Map Reduce	Pig	Hive	HBase	Storm	Spark	R Server

Core Engine

- Apache Pig

  - http://pig.apache.org/

  - Pig is a platform for analysing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelisation, which in turns enables them to handle very large data sets.

  - PIG is a data flow language. Pig manages the flow of data from input source to output store.

  - Pig provides relational (SQL) operators (JOIN, GROUP BY, etc).

  - Pig translates scripts to MapReduce tasks.
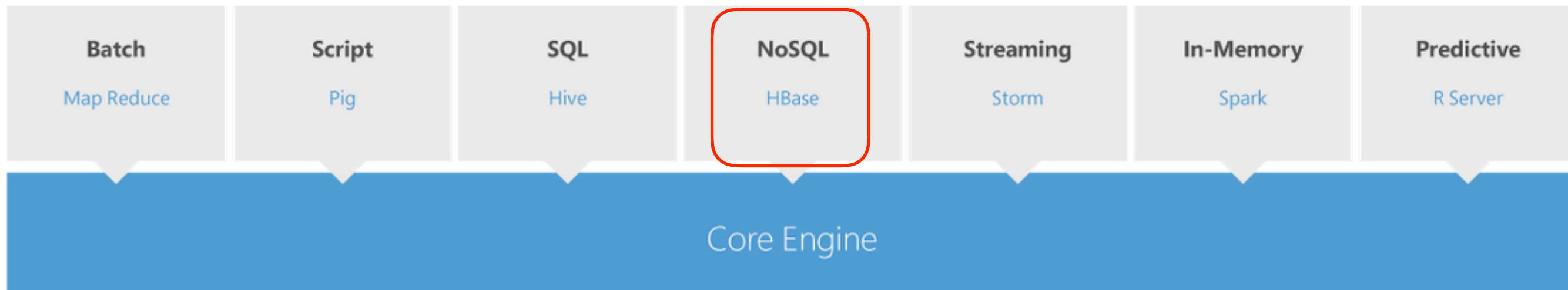
Data analytic tools

Batch	Script	SQL	NoSQL	Streaming	In-Memory	Predictive
Map Reduce	Pig	Hive	HBase	Storm	Spark	R Server

Core Engine

- Apache Hive

  - https://hive.apache.org/

  - The Apache Hive data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL. Built on top of Apache Hadoop.

  - Hive provides tools to enable easy access to data via **SQL**, thus enabling data warehousing tasks such as extract/transform/load (ETL), reporting, and data analysis.

  - Hive provides a mechanism to impose structure on a variety of data formats

  - Hive provides access to files stored either directly in Apache HDFS or in other data storage systems such as Apache HBase.

  - Hive provides query execution via Apache Tez, Apache Spark, or MapReduce.
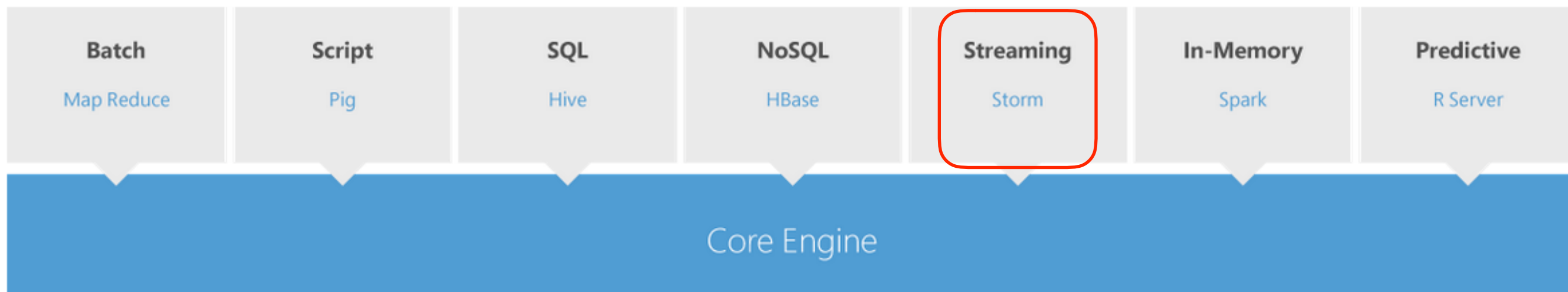
Data analytic tools

Batch	Script	SQL	NoSQL	Streaming	In-Memory	Predictive
Map Reduce	Pig	Hive	HBase	Storm	Spark	R Server

Core Engine

- Apache HBase

  - https://hbase.apache.org/

  - This project's goal is the hosting of very large tables -- billions of rows X millions of columns -- atop clusters of commodity hardware. Apache HBase is an open-source, distributed, versioned, non-relational database modelled after Google's Bigtable: A Distributed Storage System for Structured Data by Chang et al. Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

  - HBase is a type of "NoSQL" database. "NoSQL" is a general term meaning that the database isn't an RDBMS which supports SQL as its primary access language, but there are many types of NoSQL databases: BerkeleyDB is an example of a local NoSQL database, whereas HBase is very much a distributed database.

  - HDFS is a distributed file system that is well suited for the storage of large files. Its documentation states that it is not, however, a general purpose file system, and does not provide fast individual record lookups in files. HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables. This can sometimes be a point of conceptual confusion. HBase internally puts your data in indexed "StoreFiles" that exist on HDFS for high-speed lookups.
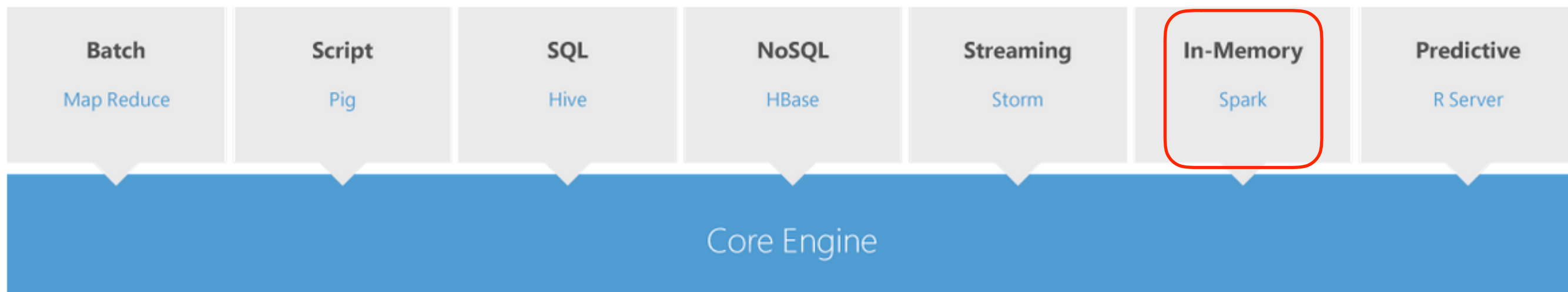
Data analytic tools

Batch	Script	SQL	NoSQL	Streaming	In-Memory	Predictive
Map Reduce	Pig	Hive	HBase	Storm	Spark	R Server

Core Engine

- Apache Storm

  - http://storm.apache.org/

  - Storm has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

  - A Storm application is designed as a "topology" in the shape of a directed acyclic graph (DAG) with spouts and bolts acting as the graph vertices. Edges on the graph are named streams and direct data from one node to another. Together, the topology acts as a data transformation pipeline.

  - At a superficial level the general topology structure is similar to a MapReduce job, with the main difference being that data is processed in real time as opposed to in individual batches. Additionally, Storm topologies run indefinitely until killed, while a MapReduce job DAG must eventually end.

Data analytic tools

Batch	Script	SQL	NoSQL	Streaming	In-Memory	Predictive
Map Reduce	Pig	Hive	HBase	Storm	Spark	R Server

Core Engine

- Apache Spark

  - http://spark.apache.org/

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

  - Apache Spark provides programmers with an application programming interface entered on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.

  - Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.

Data analytic tools

Batch	Script	SQL	NoSQL	Streaming	In-Memory	Predictive
Map Reduce	Pig	Hive	HBase	Storm	Spark	R Server

Core Engine

- R

  - https://www.r-project.org/

  - R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.

  - Can be used in conjunction with Hadoop framework, particularly Spark, to create fast, predictive analytics combined with the performance and flexibility.

Agenda

- Introduction

- MapReduce

- Big Data Infrastructure and Analytic Tools

- **Cloud and Big Data Programming - Hands on**

- Conclusion

LIVERPOOL
JOHN MOORES
UNIVERSITY

download text "cheatsheet" here:
https://github.com/posco/mmubdw

Azure - Create a Virtual Machine

- Azure Portal - https://portal.azure.com/
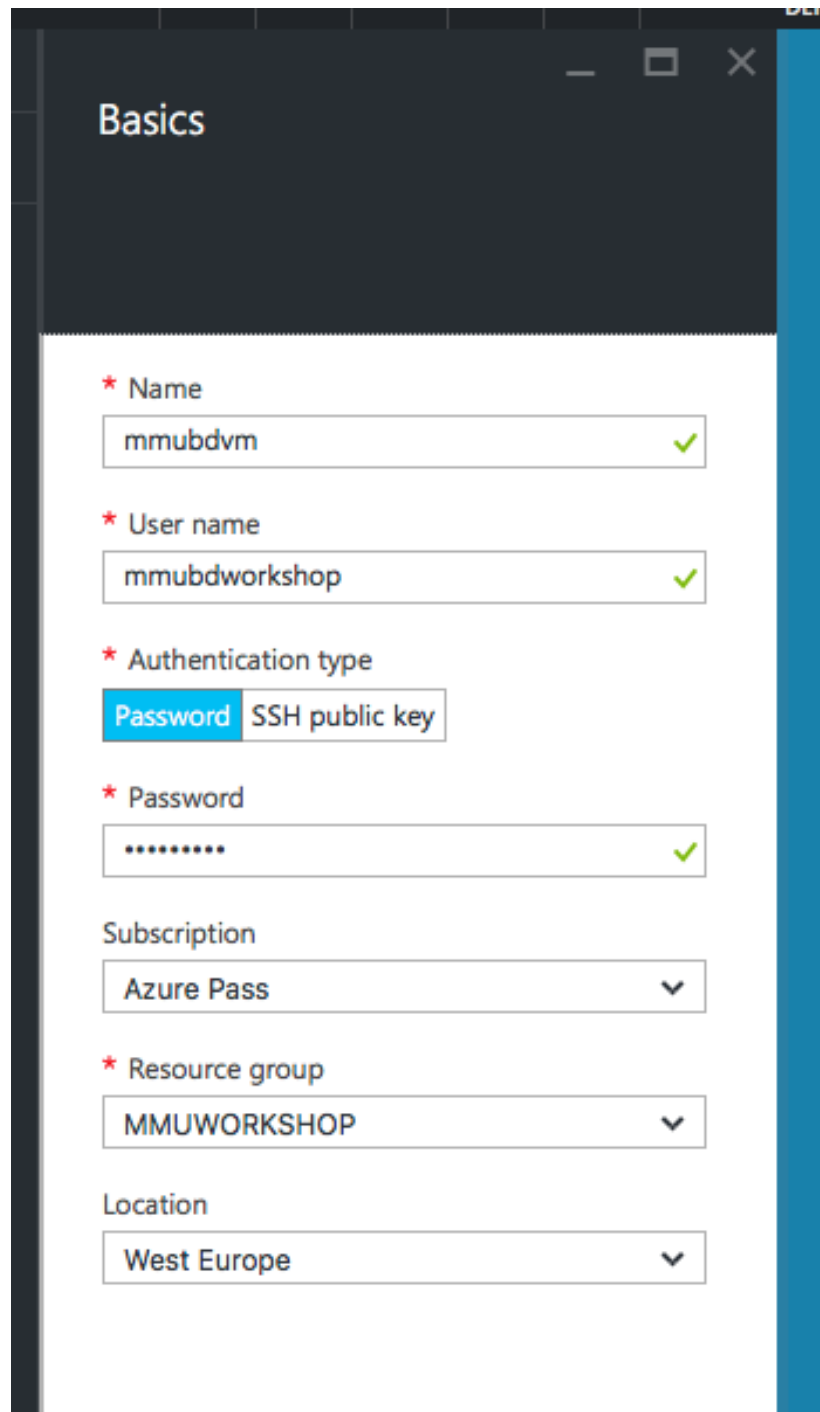
1. Click "Virtual machines" on the menu.

2. Click "+ (Add)" from the virtual machines directory.

3. Click "Ubuntu Server" recipe.

Ubuntu Server
Canonical
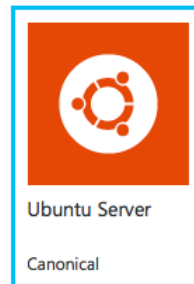
4. Choose "Ubuntu Server 16.04 LTS"

5. Click "Create" to start creating this virtual machine

6. Fill out the **Basics** form required for VM - remember your **username** and **password** as we will need them for login later.

LIVERPOOL
JOHN MOORES
UNIVERSITY

Azure - Create a Virtual Machine

7. Once you have filled out the form, click "OK" to go to next step.

8. Next, choose your favourite VM size.

9. Next, you will be asked to choose storage, use default values for all and click "OK"

10. Finally you will be brought to "Summary" page. Click "OK" to confirm creation.

11. It will take up to a few minutes for your VM to be created.

Azure - Log on to a VM

Now your VM has been created, we need to log on and use the VM.

1. Still on Azure Portal, click on "All resources", you should find all resources related to the created VM.
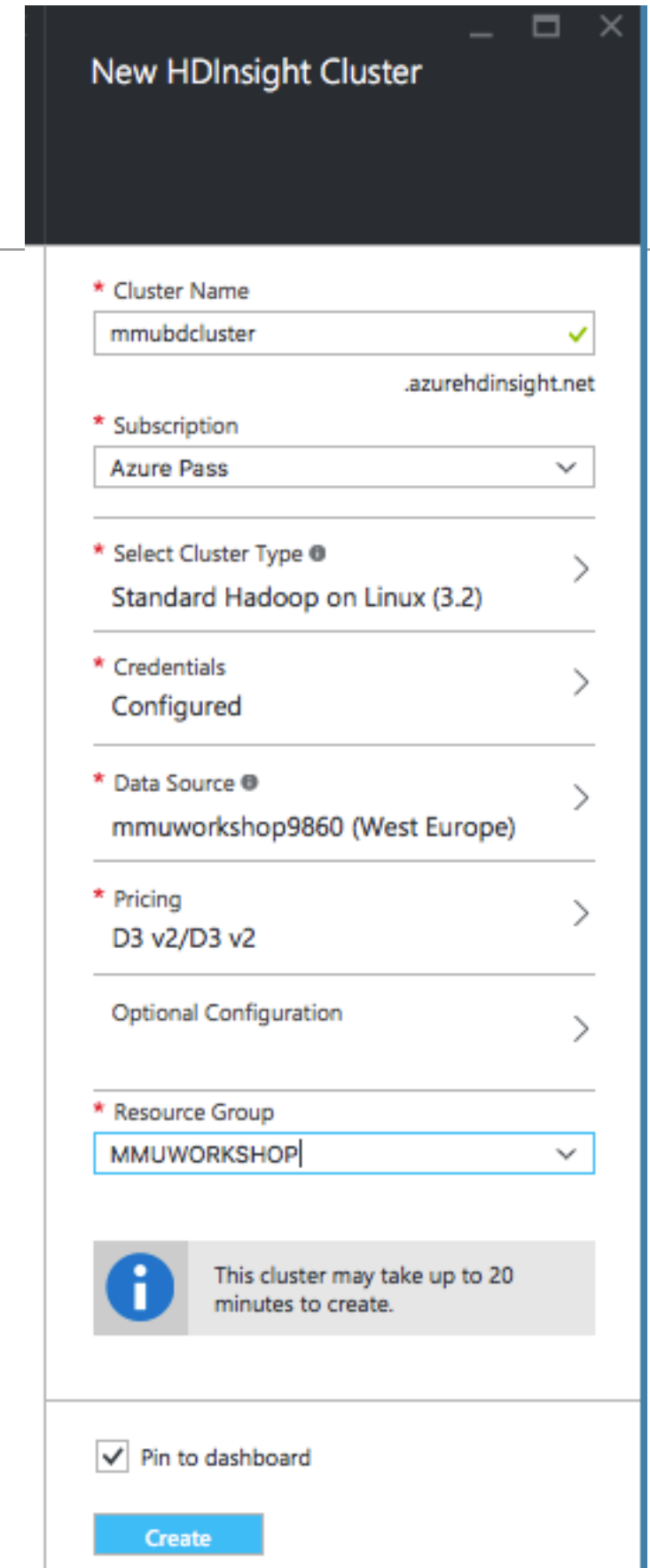
	mmubdvm	Virtual machine	MMUWORKSHOP
	mmubdvm	Network security gr...	MMUWORKSHOP
	mmubdvm	Public IP address	MMUWORKSHOP
	mmubdvm199	Network interface	MMUWORKSHOP

2. Click on "Public IP address" (the one that has a browser logo in front of it).

3. Wait until VM information is loaded, write down the "IP address". The IP looks like 13.80.64.39

4. We will use SSH, which is a secure shell service. The basic command is:
   ssh username@ip_or_domain_name_of_your_vm
   If you're using Linux/Mac, open a terminal and run the command.

✳ On Windows, you can use putty (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)

Azure - Create a HDInsight cluster

- Azure Portal - https://portal.azure.com/

  - Click "Browse" from menu, find " HDInsight Clusters"

  - Click "+ (Add)" from the "HDInsight Cluster" directory

  - Fill in the required information and click "Create"

  - It can take up to 20 minutes to create.

LIVERPOOL
JOHN MOORES
UNIVERSITY

Develop Java MapReduce programs for Hadoop

1. Prepare VM's development environment.

- Install Java JDK 8

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

- Install Apache Maven

```
sudo apt-get install maven
```

- Install a handy text editor

```
sudo apt-get install nano
```

Develop Java MapReduce programs for Hadoop

2. Configure environment variables

   i. Edit .bashrc, which will be executed upon user login.

   ```
   nano .bashrc
   ```

   ii. Move to the bottom of .bashrc file, add the following two lines.

   ```
   export JAVA_HOME=/usr/lib/jvm/java-8-oracle
   export PATH=$PATH:$JAVA_HOME/bin
   ```

3. Create a new Maven project

   i. Use the **mvn** command, which is installed with Maven, to generate the scaffolding for the project.

   ```
   mvn archetype:generate -DgroupId=org.apache.hadoop.examples -
   DartifactId=wordcountjava -DarchetypeArtifactId=maven-archetype-quickstart
   -DinteractiveMode=false
   ```

Develop Java MapReduce programs for Hadoop

This will create a new directory in the current directory, with the name specified by the artifactID parameter (wordcountjava in this example.) This directory will contain the following items:

- **pom.xml** - The Project Object Model (POM) that contains information and configuration details used to build the project.

- **src** - The directory that contains the main/java/org/apache/hadoop/examples directory, where you will author the application.

ii. Delete the **src/test/java/org/apache/hadoop/examples/apptest.java** file, as it will not be used in this example.

Develop Java MapReduce programs for Hadoop

4. Add dependencies to the project

    i.Edit the **pom.xml** file and add the following inside the <dependencies> section:

```xml
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-mapreduce-examples</artifactId>
  <version>2.5.1</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-mapreduce-client-common</artifactId>
  <version>2.5.1</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-common</artifactId>
  <version>2.5.1</version>
  <scope>provided</scope>
</dependency>
```

Develop Java MapReduce programs for Hadoop

ii. Add the following to the ***pom.xml*** file. This must be inside the <project>...</project> tags in the file; for example, between </dependencies> and </project>.

```xml
<build>
 <plugins>
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-shade-plugin</artifactId>
   <version>2.3</version>
   <configuration>
    <transformers>
     <transformer implementation="org.apache.maven.plugins.shade.resource.ApacheLicenseResourceTransformer">
     </transformer>
    </transformers>
   </configuration>
   <executions>
    <execution>
     <phase>package</phase>
      <goals>
       <goal>shade</goal>
      </goals>
    </execution>
   </executions>
  </plugin>
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-compiler-plugin</artifactId>
   <configuration>
    <source>1.7</source>
    <target>1.7</target>
   </configuration>
  </plugin>
 </plugins>
</build>
```

LIVERPOOL
JOHN MOORES
UNIVERSITY

Develop Java MapReduce programs for Hadoop

5. Create the MapReduce application

    i. Go to the **wordcountjava/src/main/java/org/apache/hadoop/examples** directory and rename the **App.java** file to **WordCount.java**.

    ii. Open the **WordCount.java** file in a text editor

        `nano WordCount.java`

    and replace the contents with the following:

```java
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Develop Java MapReduce programs for Hadoop

6. Build the application

    i. Change to the **wordcountjava** directory, if you are not already there.

    ii. Use the following command to build a JAR file containing the application:

```
mvn clean package
```

        This will clean any previous build artefacts, download any dependencies that have not already been installed, and then build and package the application. Once the command finishes, the *wordcountjava*/target directory will contain a file named ***wordcountjava-1.0-SNAPSHOT.jar***.

Develop Java MapReduce programs for Hadoop

7. Upload the app (jar)

   i. Use the following command to upload the jar file to the HDInsight headnode:

   ```
   scp wordcountjava-1.0-SNAPSHOT.jar USERNAME@CLUSTERNAME-ssh.azurehdinsight.net:
   ```

   ii. Replace __USERNAME__ with your SSH user name for the cluster. Replace __CLUSTERNAME__ with the HDInsight cluster name.

LIVERPOOL
JOHN MOORES
UNIVERSITY

Develop Java MapReduce programs for Hadoop

8. Run the MapReduce job

i. Connect to HDInsight using SSH.

ii. From the SSH session, use the following command to run the MapReduce application:

```
yarn jar wordcountjava.jar org.apache.hadoop.examples.WordCount wasb:///
example/data/gutenberg/davinci.txt wasb:///example/data/wordcountout
```

This will use the WordCount MapReduce application to count the words in the davinci.txt file, and store the results to wasb:///example/data/wordcountout. Both the input file and output are stored to the default storage for the cluster.

iii. Once the job completes, use the following to view the results:

```
hdfs dfs -cat wasb:///example/data/wordcountout/*
```

You should receive a list of words and counts.

LIVERPOOL
JOHN MOORES
UNIVERSITY

Take-home messages

- There are a plenty of big data analytic tools in the wild, pick that suits your needs best. For examples:

  - Stored data vs Streaming data

  - SQL like queries?

- Cloud can make your life easier at some little expenses.

?

p.tso@ljmu.ac.uk
@drscake
https://github.com/posco/mmubdw