Hadoop Ecosystem

Corso di Sistemi e Architetture per Big Data
A.A. 2021/22
Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

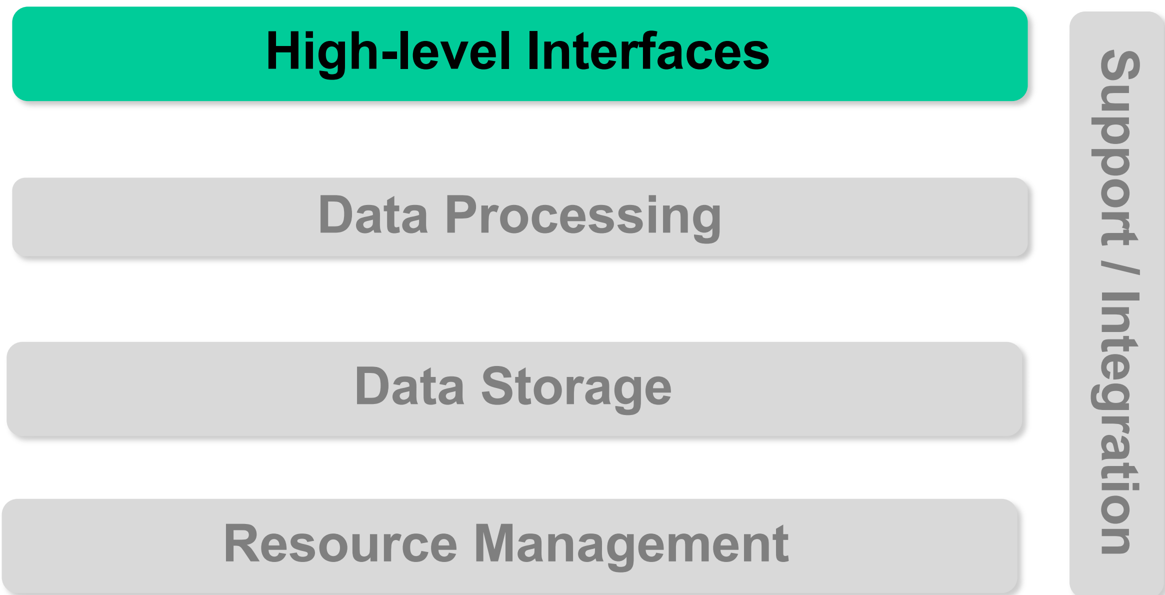
Why an ecosystem

- Apache Hadoop released in 2011
- A platform around which an entire ecosystem of capabilities has been and is built
  - Dozens of self-standing software projects (some are top projects), each addressing a variety of Big Data space and meeting different needs
- Ecosystem: complex, evolving, and not easily parceled into neat categories
- See https://hadoopecosystemtable.github.io

Some products in the ecosystem

- In this lesson
  - **Apache Pig**: high-level language plus compiler that produces sequences of MR programs
  - **Apache Hive**: SQL on top of Hadoop MapReduce
  - **Apache Impala**: distributed SQL query engine
  - **Apache Oozie**: workflow scheduler system to manage MR jobs

The reference Big Data stack



Apache Pig: motivation

- Big Data
  - 3 V, especially variety and volume
  - Most times no need to alter the original data, just to read
  - Data may be temporary; could discard dataset after analysis
- Data analysis goals
  - Quick
    - Exploit parallel processing power of a distributed system
  - Easy
    - Write a program or query without a huge learning curve
    - Have some common analysis tasks predefined
  - Flexible
    - Transforms dataset into a workable structure without much overhead
    - Performs customized processing
  - Transparent

Apache Pig: solution

- High-level data processing built on top of MapReduce which makes it easy for developers to write data analysis scripts
  - Initially developed by Yahoo!
- Scripts translated into MapReduce (MR) programs by **Pig compiler**
- Includes a high-level language (**Pig Latin**) for expressing data analysis program
- Uses MapReduce to execute all data processing
  - Compiles Pig Latin scripts written by users into a series of one or more MapReduce jobs that are then executed
- Available also on top of Spark as execution engine, but only a proof-of-concept implementation

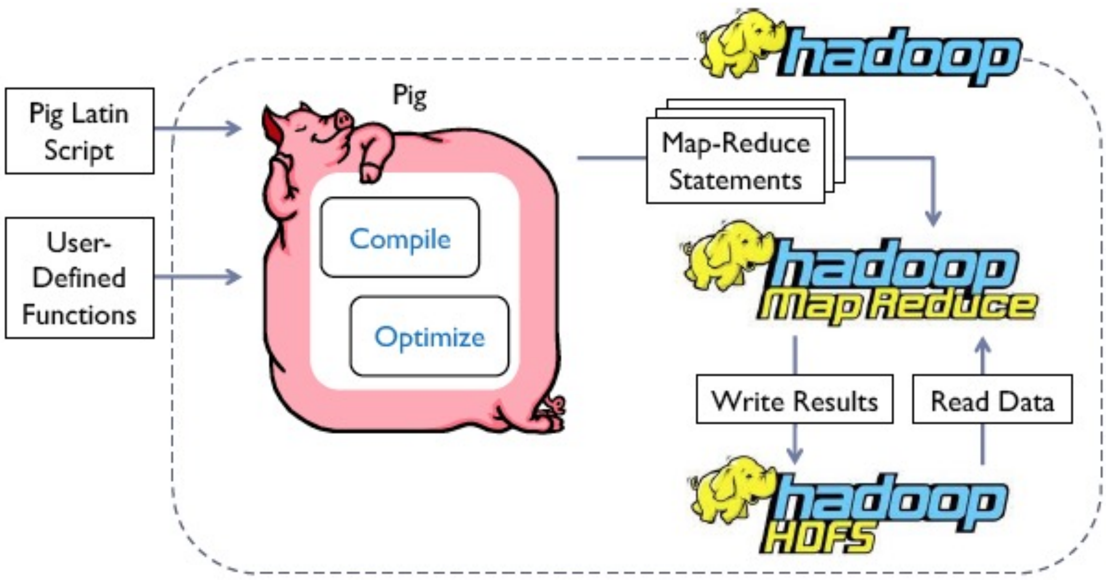
Pig Latin

- Set-oriented and procedural data transformation language
  - Primitives to filter, combine, split, and order data
  - Focus on data flow: no control flow structures like for loop or if structures
  - Users describe transformations in steps
  - Each set transformation is stateless
- Flexible data model
  - Nested bags of tuples
  - Semi-structured data types
- Executable in Hadoop
  - A compiler converts Pig Latin scripts to MapReduce jobs

Pig script compilation and execution

- Programs in Pig Latin are firstly parsed for syntactic and instance checking
  - Parser output is a logical plan, arranged in a DAG allowing logical optimizations
- Logical plan compiled by a MR compiler into a series of MR statements
- Then further optimization by a MR optimizer, which performs tasks such as early partial aggregation using MR combiners
- Finally MR program submitted to Hadoop job manager for execution

Pig: the big picture



Pig: pros

- Ease of programming
  - Complex tasks comprised of multiple interrelated data transformations encoded as data flow sequences, making them easy to write, understand, and maintain
  - Decrease in development time
- Optimization
  - The way in which tasks are encoded permits the system to optimize their execution automatically
  - Focus on semantics rather than efficiency
- Extensibility
  - Supports user-defined functions (UDFs) written in Java, Python and Javascript to do special-purpose processing

Pig: cons

- Slow start-up and clean-up of MapReduce jobs
  - It takes time for Hadoop to schedule MR jobs
- Not suitable for interactive OLAP analytics
  - When results are expected in < 1 sec
- Complex applications may require many UDFs
  - Pig loses its simplicity over MapReduce
- Debugging
  - Some produced errors caused by UDFs not helpful

Pig Latin: data model

- Atom: simple atomic value (i.e., number or string)
- Tuple: sequence of fields; each field any type
- Bag: collection of tuples
  - Duplicates are possible
  - Tuples in a bag can have different field lengths and field types
- Map: collection of key-value pairs
  - Key is an atom; value can be any type

Speaking Pig Latin

- See http://pig.apache.org/docs/r0.17.0/ for Pig Latin basics and functions

LOAD

- Input is assumed to be a bag (sequence of tuples)
- Can specify a serializer with `USING`
- Can provide a schema with `AS`

```
newBag = LOAD 'filename'
<USING functionName()>
<AS (fieldName1, fieldName2,...)>;
```


Speaking Pig Latin

FOREACH … GENERATE

- Apply specific data transformations on column data
- Each field can be:
  - A fieldname of the bag
  - A constant
  - A simple expression (i.e., f1+f2)
  - A predefined function (i.e., `SUM`, `AVG`, `COUNT`, `FLATTEN`)
  - A UDF, e.g., `tax(gross, percentage)`

```
newBag = FOREACH bagName
GENERATE field1, field2, ...;
```

- `GENERATE`: define the fields and generate a new row from the original one

Speaking Pig Latin

FILTER … BY

- Select a subset of the tuples in a bag

```
newBag = FILTER bagName BY expression;
```

- Expression uses simple comparison operators (==, !=, <, >, ...) and logical connectors (AND, NOT, OR)

```
some_apples = FILTER apples BY colour != 'red';
```

- Can use UDFs

```
some_apples = FILTER apples BY NOT isRed(colour);
```


Speaking Pig Latin

GROUP … BY

- Group together tuples having same group key

```
newBag = GROUP bagName BY expression;
```

- Usually the expression is a field

```
stat1 = GROUP students BY age;
```

- Expression can use operators

```
stat2 = GROUP employees BY salary + bonus;
```

- Can use UDFs

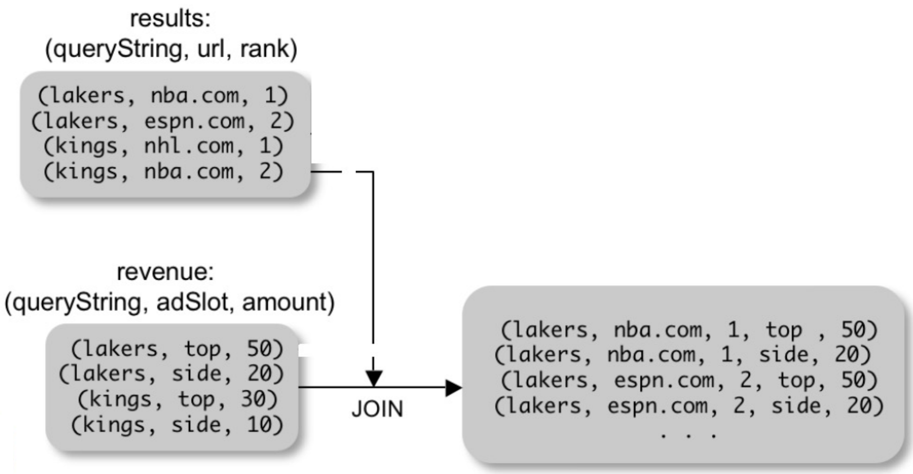
```
stat3 = GROUP employees BY netsal(salary, taxes);
```

Speaking Pig Latin

JOIN

- Join two datasets by a common field

```
joined_data = JOIN results BY queryString, revenue
BY queryString
```



Pig script for WordCount

```
data = LOAD 'input.txt' AS (line:chararray);
words = FOREACH data GENERATE FLATTEN(TOKENIZE(line)) AS word;
wordGroup = GROUP words BY word;
counts = FOREACH wordGroup GENERATE group, COUNT(words);
STORE counts INTO 'pig_wordcount';
```

- `TOKENIZE` splits a string and outputs a bag of words
- `FLATTEN` un-nests tuples as well as bags
  - The result depends on the type of structure: in the example, the bag of words is converted into tuples

Pig: how is it used in practice?

- Useful for computations across large, distributed datasets
- Abstracts away details of execution framework
- Users can change order of steps to improve performance
- Used in tandem with Hadoop and HDFS
  - Transformations converted to MapReduce data flows
  - HDFS tracks where data is stored

Hive: motivation

- Analysis of data made by both engineering and non-engineering people
- Data are growing faster and faster
  - Relational DBMS cannot handle them (e.g., limits on table size)
- Hadoop supports data-intensive distributed applications but you have to use MapReduce model
  - Hard to program
  - Not reusable
  - Error prone
  - Can require multiple stages of MapReduce jobs
  - Most users know SQL

Hive: solution

- Data warehouse built on top of Hadoop to provide data summarization, query, and analysis
  - Initially developed by Facebook
- Features
  - Makes unstructured data looks like tables
  - Queries (written in HiveQL, subset of SQL) can be directly run against these tables
  - Query execution via MapReduce
  - Access to different distributed storage (HDFS, Hbase)
- Key building principles
  - SQL is a familiar language
  - Extensibility: types, functions, formats, scripts
  - Performance

Hive: application scenario

- No real-time queries
  - Because of high latency
- No support for row-level updates
- Not suitable for OLTP
  - Lack of support for insert and update operations at row level
- Best use: batch processing over large sets of immutable data
  - Log processing
  - Data/text mining
  - Business intelligence

Hive deployment

- To deploy Hive, you also need to deploy a metastore service
  - To store the metadata for Hive tables and partitions in a RDBMS, and provides Hive access to this information

- By default, Hive records metastore information in a MySQL database on the master node's file system

Hive: example using Amazon EMR

- Launch an Amazon EMR cluster and run a Hive script to analyze a series of Amazon CloudFront access log files stored in Amazon S3

- Example of entry in log file:

```
2014-07-05 20:00:00 LHR3 4260 10.0.0.15 GET eabcd12345678.cloudfront.net
/test-image-1.jpeg 200 -
Mozilla/5.0%20(MacOS;%20U;%20Windows%20NT%205.1;%20en-
US;%20rv:1.9.0.9)%20Gecko/2009040821%20IE/3.0.9
```

Hive: example using Amazon EMR

- The Hive script:
  - Creates `cloudfront_logs` table
  - Loads the log files stored in S3 into `cloudfront_logs` table parsing the log files using the built-in regular expression serializer/deserializer (RegEx SerDe)
  - Runs a HiveQL query on the `cloudfront_logs` table to retrieve the total number of requests per operating system for a given time frame
  - Writes query result to Amazon S3 output bucket specified in the configuration
- Let's examine the Hive script

Hive: example using Amazon EMR

- Create a Hive table

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
    DateObject Date,
    Time STRING,
    Location STRING,
    Bytes INT,
    RequestIP STRING,
    Method STRING,
    Host STRING,
    Uri STRING,
    Status INT,
    Referrer STRING,
    OS String,
    Browser String,
    BrowserVersion String
)
```

Hive: example using Amazon EMR

- Load the log files stored in S3 into the `cloudfront_logs` table parsing the log files by means of the regular expression serializer/deserializer (RegEx SerDe)

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe’
WITH SERDEPROPERTIES (
 "input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^
 ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^
 ]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^\/]+)[\/](.*)$"
) LOCATION '${INPUT}/cloudfront/data';
```


Example with Amazon EMR

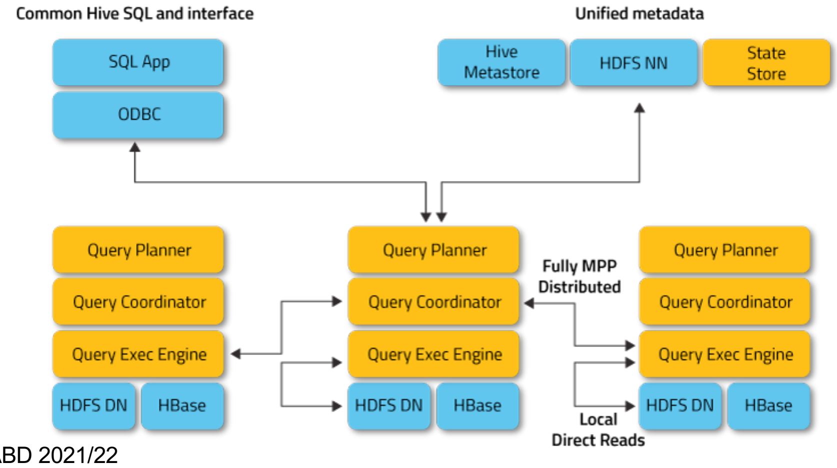
- Submit a query in HiveQL to retrieve the total number of requests per operating system for a given time frame

```
INSERT OVERWRITE DIRECTORY '${OUTPUT}/os_requests/'
SELECT os, COUNT(*) count FROM cloudfront_logs
WHERE dateobject BETWEEN '2014-07-05' AND '2014-08-
05' GROUP BY os;
```

  - `INSERT OVERWRITE DIRECTORY` overwrites the existing data in the directory with the new values using Hive SerDe. The inserted rows can be specified by result from a query or value expressions

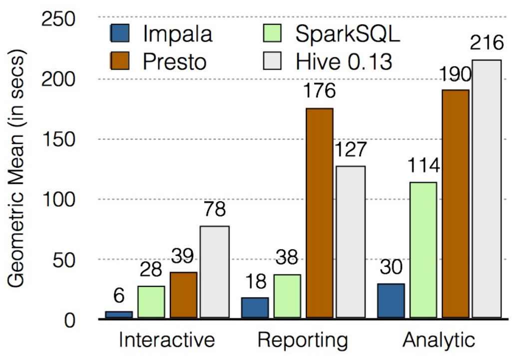
Apache Impala

- Distributed SQL query engine for petabytes of data stored in Apache Hadoop clusters
  - Based on scalable parallel database technology
  - Inspired by Google Dremel
  - Provides low latency and high concurrency for BI/analytic queries (not delivered by batch frameworks such as Hive)
  - Input data can be stored in HDFS or HBase

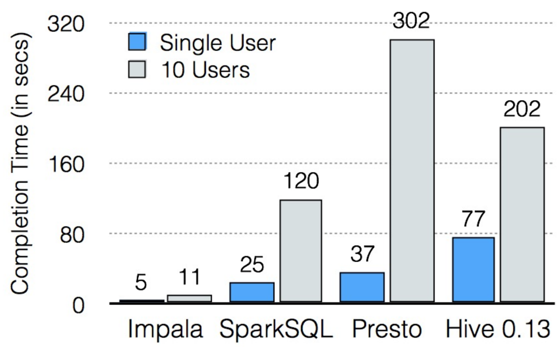


Impala: performance

- Performance: one order of magnitude faster than Hive, significantly faster than Spark SQL (in 2015)



Single user



Multiple users

[Impala: A Modern, Open-Source SQL Engine for Hadoop]

Managing complex jobs

- How to simplify the management of complex Hadoop jobs?
- How to manage a recurring query?
  - i.e., a query that repeats periodically
  - Naïve approach: manually re-issue the query every time it needs to be executed
    - Lacks convenience and system-level optimizations

Apache Oozie

- Workflow scheduler system to manage Apache Hadoop jobs
- Java web app that runs in a Java servlet-container
- Integrated with Hadoop ecosystem, supports different types of jobs
  - E.g., Hadoop MapReduce, Pig, Hive

Oozie: workflow

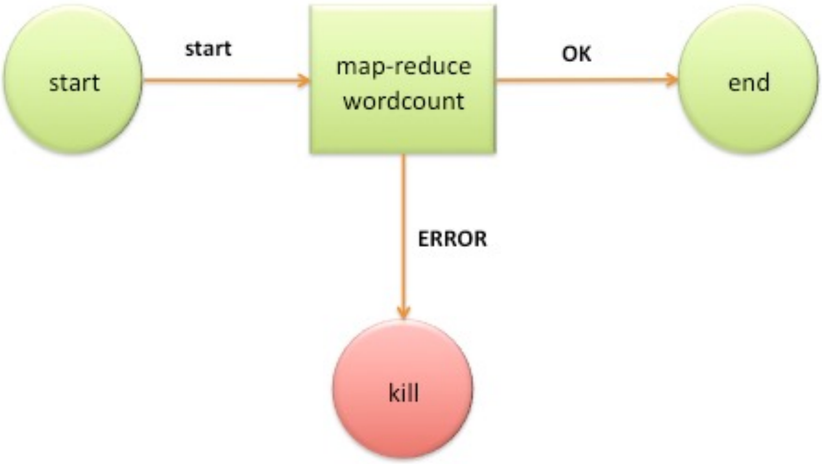
- Workflow: collection of actions (e.g., MapReduce jobs, Pig jobs) arranged in a control dependency DAG
  - Control dependency from one action to another means that the second action can't run until the first action has completed
- Workflow definition written in hPDL
  - A XML Process Definition Language

Oozie: workflow

- Control flow nodes
  - Define beginning and end of workflow (start, end and fail nodes)
  - Provide a mechanism to control the workflow execution path (decision, fork and join)
- Action nodes
  - Mechanism by which a workflow triggers the execution of a computation/processing task
  - Can be extended to support additional type of actions
- Oozie workflows can be parameterized using variables like ${inputDir} within the workflow definition
  - If properly parameterized (i.e. using different output directories) several identical workflow jobs can concurrently run

Oozie: workflow example

- Example of Oozie workflow: Wordcount



Oozie: workflow example

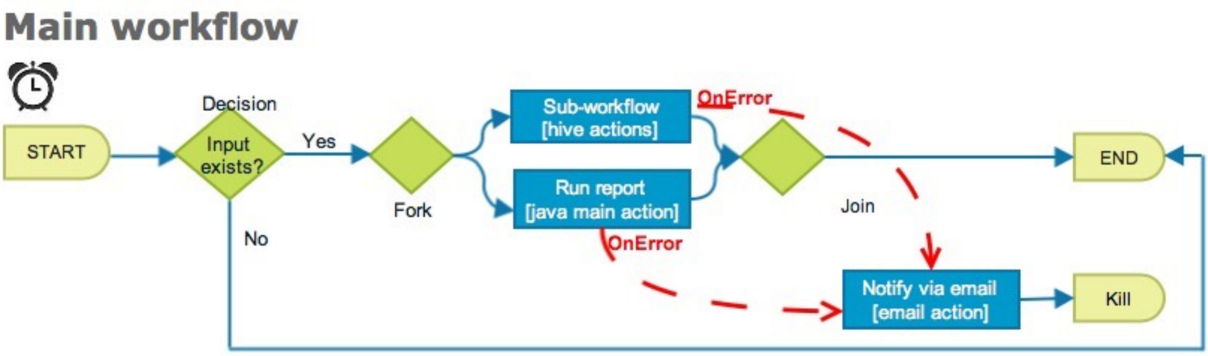
```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
    <start to='wordcount'/>
    <action name='wordcount'>
        <map-reduce>
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.mapper.class</name>
                    <value>org.myorg.WordCount.Map</value>
                </property>
                <property>
                    <name>mapred.reducer.class</name>
                    <value>org.myorg.WordCount.Reduce</value>
                </property>
                <property>
                    <name>mapred.input.dir</name>
                    <value>${inputDir}</value>
                </property>
```

Oozie: workflow example

```
                <property>
                    <name>mapred.output.dir</name>
                    <value>${outputDir}</value>
                </property>
            </configuration>
        </map-reduce>
        <ok to='end'/>
        <error to='end'/>
    </action>
    <kill name='kill'>
        <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
    </kill/>
    <end name='end'/>
</workflow-app>
```

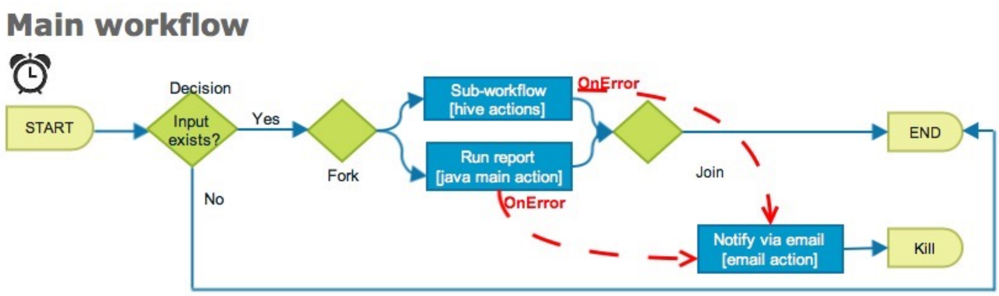

Oozie: fork and join

- A fork node splits one path of execution into multiple concurrent paths of execution
- A join node waits until every concurrent execution path of a previous fork node arrives to it
- Fork and join must be used in pairs



Oozie: coordinator

- Workflow jobs can be run based on regular time intervals and/or data availability or can be triggered by some external event
- Oozie coordinator allows the user to define and execute recurrent and interdependent workflow jobs
  - Triggered by time (frequency) and data availability



References

- Gates et al., Building a high-level dataflow system on top of Map-Reduce: the Pig experience, *Proc. VLDB Endow.,* 2009.
- Thusoo et al., A petabyte scale data warehouse using Hadoop, *IEEE ICDE '10*, 2010.
- Kornacker et al., Impala: A Modern, Open-Source SQL Engine for Hadoop, *CIDR '15*, 2015.