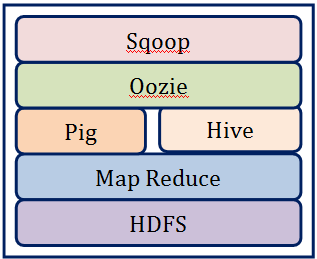
The complete structure and the working of “Oozie Workflow scheduler “:

**Introduction:**  
  
In Hadoop Ecosystem, most of the functionality like map-reduce jobs, pig scripts and hive queries are executed as batch jobs.

This creates a lot of overhead in deployment and maintenance of hadoop components.

As a solution to this, Oozie provides workflows in xml format using which we can define multiple Map/Reduce jobs into a logical unit of work, accomplishing the larger task.

This helps in chaining the related MapReduce jobs which can be either Hive queries or Pig scripts like mentioned in below diagram.



**Oozie Workflow scheduler:**

An Oozie workflow is a collection of actions arranged in a directed acyclic graph (DAG). This graph can contain two types of nodes:

**control nodes**

**Action nodes**.

**Control nodes,** which are used to define job chronology, provide the rules for beginning and ending a workflow and control the workflow execution path with possible decision points known as fork and join nodes.

**Action nodes** are used to trigger the execution of tasks. In particular, an action node can be a MapReduce job, a Pig application, a file system task, or a Java application. (The shell and ssh actions have been deprecated).

Oozie is a native Hadoop stack integration that supports all types of Hadoop jobs and is integrated with the Hadoop stack. In particular, Oozie is responsible for triggering the workflow actions, while the actual execution of the tasks is done using Hadoop MapReduce.

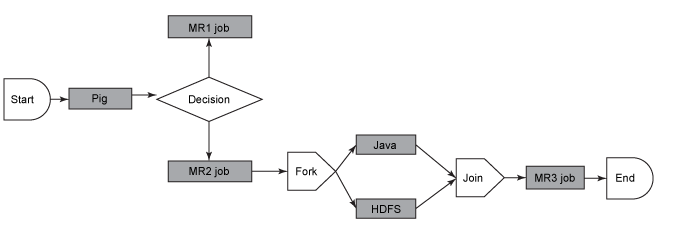
Therefore, Oozie becomes able to leverage existing Hadoop machinery for load balancing, fail-over, etc. Oozie detects completion of tasks through callback and polling.

When Oozie starts a task, it provides a unique callback HTTP URL to the task, and notifies that URL when it is complete.

If the task fails to invoke the callback URL, Oozie can poll the task for completion. Figure 1 illustrates a sample Oozie workflow that combines six action nodes (Pig scrip, MapReduce jobs, Java code, and HDFS task) and five control nodes (Start, Decision control, Fork, Join, and End).

Oozie workflows can be also parameterized. When submitting a workflow job, values for the parameters must be provided.

If the appropriate parameters are used, several identical workflow jobs can occur concurrently.



Oozie workflow definition language is XML-based and it is called the Hadoop Process Definition Language.

Oozie comes with a command-line program for submitting jobs. This command-line program interacts with the Oozie server using REST.

To submit or run a job using the Oozie client, give Oozie the full path to your workflow.xml file in HDFS as a parameter to the client.

Oozie does not have a notion of global properties. All properties, including the jobtracker and the namenode, must be submitted as part of every job run.

Oozie uses an RDBMS for storing state.

**Example of Oozie workflow:**

<workflow-app xmlns='uri:oozie:workflow:0.1' name='SimpleWorkflow'>

    <start to='ingestor'/>

    <action name='ingestor'>

        </java>

            <job-tracker>${jobTracker}</job-tracker>

            <name-node>${nameNode}</name-node>

            <configuration>

                <property>

                    <name>mapred.job.queue.name</name>

                    <value>default</value>

                </property>

            </configuration>

            <arg>${driveID}</arg>

        </java>

        <ok to='merging'/>

        <error to='fail'/>

    </action>

    <fork name='merging'>

        <path start='mergeT1'/>

        <path start='mergeT2'/>

    </fork>

    <action name='mergeT1'>

        <java>

            <job-tracker>${jobTracker}</job-tracker>

            <name-node>${nameNode}</name-node>

            <configuration>

                <property>

                    <name>mapred.job.queue.name</name>

                    <value>default</value>

                </property>

            </configuration>

            <arg>-drive</arg>

            <arg>${driveID}</arg>

            <arg>-type</arg>

            <arg>T1</arg>

        </java>

        <ok to='completed'/>

        <error to='fail'/>

    </action>

    <action name='mergeT2'>

        <java>

            <job-tracker>${jobTracker}</job-tracker>

            <name-node>${nameNode}</name-node>

            <configuration>

                <property>

                    <name>mapred.job.queue.name</name>

                    <value>default</value>

                </property>

            </configuration>

            <main-class>com.navteq.assetmgmt.hdfs.merge.MergerLoader</main-class>

            <arg>-drive</arg>

            <arg>${driveID}</arg>

            <arg>-type</arg>

            <arg>T2</arg>

        </java>

        <ok to='completed'/>

        <error to='fail'/>

    </action>

    <join name='completed' to='end'/>

    <kill name='fail'>

        <message>Java failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>

    </kill>

    <end name='end'/>

</workflow-app>

**This simple workflow defines three actions:**

**ingestor,**

**mergeT1, and**

**mergeT2.**

Each action is implemented as a MapReduce job,the workflow starts with the start node, which transfers control to the ingestor action.

Once the ingestor step completes, a fork control node is invoked, an action that starts the execution of mergeT1 and mergeT2 in parallel.

Once both actions are completed, the join control node is invoked. On successful completion of join node, the control is passed to the end node, a step that ends the process.

The <job-tracker> and <name-node> entities dictate the servers that the Hive job will connect to for executing its script.

