Strategies for scheduling Hadoop Jobs

Pere Urbon-Bayes (@purbon)
pere.urbon@gmail.com
http://www.purbon.com
$ whoami

Software Architect with > 10 years of experience. Interested in data centric applications and graph analytics.
I run the GraphDevRoom at FOSDEM.
Movie and Series Geek, Runner, doing everything I can to enjoy this life.

I believe we're getting payed for solving problems, not for having strong religious fights between technologies.
Houston, we've had a problem!
An state of the art.
How to approach an internal solution.
Houston, we've had a problem!.

An state of the art.

How to approach an internal solution.

Scheduler.register(talk, Time.now)
The situation

- We build large solar fields, for example in Templin, the north of Berlin, DE.
  - 214 hectares
  - 1,500,000 photovoltaic modules
  - 128.48 MW
  - 205 millions Euro

- Inside knowledge is very much appreciated.
A platform for this?

The scheduler

NoSQL Matters Cologne 2014
The scheduler

A nice scheduler should:
- Run hadoop jobs, but also PIG or shell scripts, etc.
- Consider executions idempotent.
- Have a nice interaction way, for example a REST api.
- Support workflows but also cron alike jobs.
- Track the job execution.
- Provide live statistics for monitoring.
Scheduler.register(talk, Time.now)

Houston, we've had a problem!

An state of the art.

How to approach an internal solution.
State of the art

NoSQL Matters Cologne 2014
State of the art

http://oozie.apache.org/

NoSQL Matters Cologne 2014
State of the art

Azkaban
Open-source Workflow Manager

MapReduce
JobTracker
TaskTracker

Database

Azkaban Server
Azkaban Executor
“Client”

http://azkaban.github.io/azkaban2/

NoSQL Matters Cologne 2014
State of the art

Luigi Worker Pool

Luigi scheduler

HDFS

MapReduce
  JobTracker
  TaskTracker

Deployment throw codebase copy

Client

https://github.com/spotify/luigi
# State of the art

<table>
<thead>
<tr>
<th></th>
<th>Lang</th>
<th>Complexity (LOC)</th>
<th>Framework</th>
<th>Logs</th>
<th>Community</th>
<th>Docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>oozie</td>
<td>java</td>
<td>High &gt; 105k</td>
<td>Pig, hive, sqoop, mapreduce</td>
<td>decentralized</td>
<td>Good - ASF</td>
<td>excellent</td>
</tr>
<tr>
<td>azkaban</td>
<td>java</td>
<td>Moderate &gt; 26k</td>
<td>Pig, hive, mapreduce</td>
<td>Centralized</td>
<td>Few users</td>
<td>good</td>
</tr>
<tr>
<td>luigi</td>
<td>phython</td>
<td>Simple &gt; 5.9k</td>
<td>Hive, postgress, scalding, python, streaming</td>
<td>decentralized</td>
<td>Few users</td>
<td>good</td>
</tr>
</tbody>
</table>

http://www.slideshare.net/jcrobak/data-engineermeetup-201309
# State of the art

<table>
<thead>
<tr>
<th></th>
<th>Configuration</th>
<th>Replay</th>
<th>Customization</th>
<th>Testing</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>oozie</strong></td>
<td>Command line, property files, xml</td>
<td>Oozie job - rerun</td>
<td>difficult</td>
<td>Mini oozie</td>
<td>Kerberos, simple, custom</td>
</tr>
<tr>
<td><strong>azkaban</strong></td>
<td>Bundled inside the workflow zip, systems defaults</td>
<td>Partial rerun UI</td>
<td>plugin</td>
<td>?</td>
<td>Xml based, custom</td>
</tr>
<tr>
<td><strong>luigi</strong></td>
<td>Command line, python init file</td>
<td>Remove output - idempotent</td>
<td>subclass</td>
<td>Python unit test</td>
<td>Linux, OS based</td>
</tr>
</tbody>
</table>

http://www.slideshare.net/jcrobak/data-engineermeetup-201309
State of the art

So why not jumping into Oozie, it have:

• Great community and features
• A complete integration with the common Hadoop tooling
• Has a good documentation
State of the art

But, it also have:

- An XML based configuration
- Scheduling based on map tasks
- A Hard setup
- An obscure object model

*And in this project case, we don't like plain Java that much.*
Scheduler.register(talk, Time.now)

Houston, we've had a problem!

An state of the art.

How to approach an internal solution.
The wish list

- Easy to interact with, ex. REST api.
- Manage workflows and simple cron alike jobs.
- Have a centralized logging infrastructure.
- To run not just Hadoop jobs, but also support scripting.
- Easy configuration and deployment model.
- Simplified extension model.
The wish list

- Be able to reply jobs.
  - This means jobs data to be idempotent and status management.
- A report and monitoring module.
- Track the job execution.
  - This includes failures, timing, memory consumption, etc.

*But, for sure they are not all necessary at once*
Our friends of convenience

Sinatra  Neo4j  JRuby  Hadoop

NoSQL Matters Cologne 2014
A possible architecture?

- Scheduler Server
- Registry
- Cron Manager
- Workflow Manager
- Scheduler Worker
- Database
- GraphDB
- File System
- MapReduce JobTracker TaskTracker
- REST api
- NoSQL Matters Cologne 2014
The workflow graph

defined as Direct Acyclic Graph (DAG)
Hadoop (First approach)

```java
public void run(String job, Map<String, Object> options) throws IOException {
    Path input = new Path("path/to/the/main/input/dataset");
    Path output = new Path("path/to/the/main/output/dataset");
    
    if ((Boolean) options.get("pre")) {
        Path preInput = new Path("path/to/the/input/dataset");
        Path preOutput = new Path("path/to/the/output/dataset");
        JobConf preJob = new CleansingJob(conf, preInput, preOutput, options);
        preInput = new Path("path/to/the/input/dataset");
        preOutput = new Path("path/to/the/output/dataset");
        JobConf jobConf = new AggregationJob(conf, preInput, preOutput);
        JobClient.runJob(preJob);
        JobClient.runJob(jobConf);
    }
    JobConf mainConf = new MainJob(conf, input, output, options);
    JobClient.runJob(mainConf);
}
```

Submit and run tasks one after another.
Hadoop (second approach)

```java
public RunningJob run(String jobcode, List<Path> inputs, Path output, Map
RunningJob job = null;
if (JDL_JOB_CODE.equalsIgnoreCase(jobcode)) {
    conf.setMapOutputValueClass(DoubleArrayWritable.class);
    Path input = inputs.get(0);
    JobConf mainConf = createJDLJob(conf, input, output, options);
    JobClient client = new JobClient(mainConf);
    job = client.submitJob(mainConf);
} else if (SMART_JOB_CODE.equalsIgnoreCase(jobcode)) {
    JobClient client = new JobClient(conf);
    job = client.submitJob(conf);
} else if (DIFF_JOB_CODE.equalsIgnoreCase(jobcode)) {
    Path input = inputs.get(0);
    JobConf postConf = createJDLJob(conf, input, output, options);
    JobClient client = new JobClient(postConf);
    job = client.submitJob(postConf);
} else {
    throw new IllegalArgumentException("Job not found!");
}
```
JobClient.runJob

```java
runJob

public static RunningJob runJob(JobConf job)
    throws IOException

Utility that submits a job, then polls for progress until the job is complete.

Parameters:
    job - the job configuration.

Throws:
    IOException - if the job fails
```
JobClient.submitJob

submitJob

public RunningJob submitJob(JobConf conf)
throws FileNotFoundException,
IOException

Submit a job to the MR system. This returns a handle to the RunningJob which can be used to track the running-job.

Parameters:
    conf - the job configuration.

Returns:
    a handle to the RunningJob which can be used to track the running-job.

Throws:
    FileNotFoundException
    IOException
setJobEndNotificationURI

public void setJobEndNotificationURI(String uri)

Set the uri to be invoked in-order to send a notification after the job has completed (success/failure).

The uri can contain 2 special parameters: $jobid and $jobstatus. Those, if present, are replaced by the job's identifier and completion-status respectively.

This is typically used by application-writers to implement chaining of Map-Reduce jobs in an asynchronous manner.

Parameters:
- uri - the job end notification uri

See Also:
- JobStatus, Job Completion and Chaining
Remote Job

```java
config.addResource(new Path("/usr/local/hadoop/conf/core-site.xml"));
config.addResource(new Path("/usr/local/hadoop/conf/hdfs-site.xml"));
config.set("fs.default.name", "hdfs://" + args[2] + ":54310");

JobConf job = new JobConf(config);
job.setJarByClass(MyRemoteJob.class);
```
RemoteJob

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>192.168.102.131:9000</value>
  </property>
</configuration>
```

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>192.168.102.131:9001</value>
  </property>
</configuration>
```
def add_job(pattern, collection, name, options)
    job_options = { :allow_overlapping => false, :job => true }
    job_options[:last_job_id] = options[:last_job_id] if options[:last_job_id]
    job = @scheduler.cron pattern, job_options do |job|
        run_job_now collection, name, job.job_id, options
    end
    @inspector.add_job(collection, name, job.job_id)
end
def run_job_now(collection, name, job_id, options)
    begin
        run_job "#{collection}.#{name}", options, job_id
        unless options['after'].nil? then
            options['after'].split('#').each do |conf|
                run_job conf, options, job_id
            end
        end
    end
    rescue Exception => e
        log "exception while running #{collection} #{name} #{job_id}"
        log "#{e}"
    end
end
Scheduler::Internals.run_java

def run_job(conf, options, job_id)
    time = Time.now
    begin
        collection = conf.split('.')[0]
        name = conf.split('.')[1]
        log "running #{collection} #{name} with options #{options.inspect} and conf #{conf}"
        get_clazz(collection, name).run(options)
        @inspector.add_field(collection, name, job_id, 'ok', 1)
    rescue Exception => e
        log "exception while running #{conf}"
        log "#{e}"
        @inspector.add_field(collection, name, job_id, 'ko', 1)
    end
    time = Time.now-time;
    @inspector.add_execution_time(collection, name, job_id, time)
end
Houston, we've had a problem!
An state of the art.
How to approach an internal solution.
Strategies for scheduling Hadoop jobs

Thanks for your attention! Questions?

Pere Urbon-Bayes
pere.urbon@gmail.com
http://www.purbon.com