Cloud IRAD

## **Overview**

Cloud Computing Data Processing (CCDP) is an IRAD whose intention is to create a framework to easily interact with the most common services offered by the cloud provider for data processing. The framework takes the burden away from knowledge about the cloud provider from the processing modules. It aims to facilitate an environment that is dynamically modified be either allocating or deallocating resources as needed. This dynamic environment will allow maximizing the resources usage without compromising processing threads. A processing thread is a sequence of processing modules used to generate some results. The framework provides a way to allow communications between the modules

Need to provide a way to register a processing thread or system by

* Configuration: For the thread as a whole and for each component
* Registration: That means we need a manager responsible for managing available workers. It could be self-discovered

.

CCDP can be divided into three main components:

### **Command and Control of the Modules**

This represents the program whose main function is to assign tasks to the registered processing modules. In order to achieve this we will be using Celery as the tasking queue. Each module should contain some basic configuration such as min and max number of modules required for each processing thread. The framework would allocate as many modules in a single VM as permissible to expedite processing unless the module has a specific function that could be satisfied by one of the cloud services.

### **API at the module level**

The modules are the components used to process data. The term “process data” is open to each system’s need. Regardless of the processing each module will need a way to provide input and output data. The framework will take advantage of some of the services such as DataPipeline to accommodate the modules in this regards.

Types of modules:

* Data generators: Should take advantage of AWS services such as RDS or DynamoDB at least to start
* Data processors: Shall use Data Pipeline to move data among all the workers and Machine Learning, Hadoop and Elastic Map Reduce services ([What is EMR](http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-what-is-emr.html)) and [Kinesis](https://aws.amazon.com/kinesis/?nc2=h_l3_al) for Streaming data
* Data writers: Same usage of the database nodes
* Data visualization:

### **Tools and API used to talk to the AWS services**

This is the hard part to do as this is the area we know the least. These set of tools is what will hide the services from the modules.

**Tasks**

* Create some workflow to understand how the components work as well as how their API is
* Components:
  + RDS: Get data from a database and inject it into the Data Pipeline
  + Data Pipeline: Moves the data across workers, what is the format?
  + EC2 and EC2 Containers: Need to learn how to start and stop VM as well as containers. During initial surge, low threshold, start VM and after up threshold start containers for data processing?
  + How can we take advantage of the Machine Learning capabilities?
  + Can we use ML to predict VM requirements? In other words, using ML to predict when we need to start stop VMs and containers
* Understand RedHawk GPP to determine if it should be a container or its own VM. I think containers would be better as I am presuming they are quicker to launch

Products Available on the Dark Side

* Amazon EC2
* Amazon DynamoDB
* Amazon VPC
* Amazon S3
* Amazon EBS
* Elastic Load Balancing
* Auto Scaling
* Amazon Redshift
* Amazon CloudWatch
* Amazon IAM
* AWS Direct Connect
* Amazon SQS
* Amazon RDS
* Amazon SWF
* Amazon EMR
* Amazon Cloud Trial
* Amazon CloudFormation
* Amazon CloudHSM

Tasks

* Familiarize with AWS
* Find tutorials and Examples
* Do I really need Celery? I can use it to drop files and to find modules

Modules

* Needs to be configurable
* Register with CCPD-CM
* System starts
* Modules gets data and uses DataPipeline to send it as available
* Module 2 gets stream of data

GUI shows systems and modules

* Modules can be added and removed
* double click on a module opens config
* Configuration is json
  + init, start, stop, pause
* min, max number of modules
* min, max resources to spawn another

AWS Random Notes

Following the Python API: <http://boto3.readthedocs.org/en/latest/guide/migrationec2.html>

* Can have the engine running outside of AWS and interacting with it
* Simple Storage Solution (S3) allows you to create a ‘bucket’ and in it you can upload files. Need to offer something to interface with this and being able to publish or upload files and maybe even a stream of data
* I was able to launch an image as long as it was Windows. Kept getting an error message about not being able to use Hardware Virtual Machine (HVM) on a Non-Windows Image, what the hell?

Other Random Notes

* [Python Web Frameworks](https://www.airpair.com/python/posts/django-flask-pyramid)

Hadoop Notes

* Make sure all the values inside the tags do not have empty spaces
* Edit the core-site.xml, hdfs-site.xml, and the hadoop-env.sh to run hdfs
* Need to set the input and output directory and run start-dfs.sh
* Browse the files at NameNode - http://localhost:50070/
* To remove the warning “Unable to load Hadoop libraries” do what these links suggests:
  + <http://www.ercoppa.org/Linux-Compile-Hadoop-220-fix-Unable-to-load-native-hadoop-library.htm>
    - Skip javadoc by adding -Dmaven.javadoc.skip=true
  + <http://www.ercoppa.org/Linux-Compile-Hadoop-220-fix-Unable-to-load-native-hadoop-library.htm>

Links

QuickStart: <http://spark.apache.org/docs/latest/quick-start.html>

spark-submit: <http://spark.apache.org/docs/latest/submitting-applications.html>

Cluster Mode Overview: <http://spark.apache.org/docs/latest/cluster-overview.html>

Spark:

* **Spark Streaming:**
  + [Spark Streaming](https://spark.apache.org/streaming/) can be used for processing the real-time streaming data. This is based on micro batch style of computing and processing. It uses the DStream which is basically a series of RDDs, to process the real-time data.
* **Spark SQL:**
  + [Spark SQL](https://spark.apache.org/sql/) provides the capability to expose the Spark datasets over JDBC API and allow running the SQL like queries on Spark data using traditional BI and visualization tools. Spark SQL allows the users to ETL their data from different formats it’s currently in (like JSON, Parquet, a Database), transform it, and expose it for ad-hoc querying.
* **Spark MLlib:**
  + [MLlib](https://spark.apache.org/mllib/) is Spark’s scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives.
* **Spark GraphX:**
  + [GraphX](https://spark.apache.org/graphx/) is the new (alpha) Spark API for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing the Resilient Distributed Property Graph: a directed multi-graph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and aggregateMessages) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.
* Mesos and YARN are Distributed Computing Framework
* Running multiple python files in Spark use .zip files as in spark-submit --py-files fname.zip,fname2.zip,fname3.py
* The Deploy mode can be client or cluster:
  + A common deployment strategy is to submit your application from a gateway machine that is physically co-located with your worker machines (e.g. Master node in a standalone EC2 cluster). In this setup, client mode is appropriate. In client mode, the driver is launched directly within thespark-submit process which acts as a *client* to the cluster. The input and output of the application is attached to the console. Thus, this mode is especially suitable for applications that involve the REPL (e.g. Spark shell).
  + Alternatively, if your application is submitted from a machine far from the worker machines (e.g. locally on your laptop), it is common to use clustermode to minimize network latency between the drivers and the executors. Note that cluster mode is currently not supported for Mesos clusters. Currently only YARN supports cluster mode for Python applications.

Spark Deployment Examples:

*# Run application locally on 8 cores*  
./bin/spark-submit **\**  
 --class org.apache.spark.examples.SparkPi **\**  
 --master local[8] **\**  
 /path/to/examples.jar **\**  
 100  
  
*# Run on a Spark standalone cluster in client deploy mode*  
./bin/spark-submit **\**  
 --class org.apache.spark.examples.SparkPi **\**  
 --master spark://207.184.161.138:7077 **\**  
 --executor-memory 20G **\**  
 --total-executor-cores 100 **\**  
 /path/to/examples.jar **\**  
 1000  
  
*# Run on a Spark standalone cluster in cluster deploy mode with supervise*  
./bin/spark-submit **\**  
 --class org.apache.spark.examples.SparkPi **\**  
 --master spark://207.184.161.138:7077 **\**  
 --deploy-mode cluster  
 --supervise  
 --executor-memory 20G **\**  
 --total-executor-cores 100 **\**  
 /path/to/examples.jar **\**  
 1000  
  
*# Run on a YARN cluster*  
export HADOOP\_CONF\_DIR=XXX  
./bin/spark-submit **\**  
 --class org.apache.spark.examples.SparkPi **\**  
 --master yarn **\**  
 --deploy-mode cluster **\** *# can be client for client mode*  
 --executor-memory 20G **\**  
 --num-executors 50 **\**  
 /path/to/examples.jar **\**  
 1000  
  
*# Run a Python application on a Spark standalone cluster*  
./bin/spark-submit **\**  
 --master spark://207.184.161.138:7077 **\**  
 examples/src/main/python/pi.py **\**  
 1000  
  
*# Run on a Mesos cluster in cluster deploy mode with supervise*  
./bin/spark-submit **\**  
 --class org.apache.spark.examples.SparkPi **\**  
 --master mesos://207.184.161.138:7077 **\**  
 --deploy-mode cluster  
 --supervise  
 --executor-memory 20G **\**  
 --total-executor-cores 100 **\**  
 http://path/to/examples.jar **\**  
 1000

* Spark loads the configuration from $SPARK\_HOME/conf/spark-defaults.conf

Mike TODO

* Find an EMR example
* Create two or more modules
* Each module is a separate zip file (or the same, but a zip)
* The data could come either from our own DB or external

Oscar TODO

* Create an instance for hadoop cluster
* Create a hadoop headnode
* Create an engine to read a json file with the chain
* Load the modules and modify the PYTHONPATH
* Add CloudWatch and launch more instances as needed