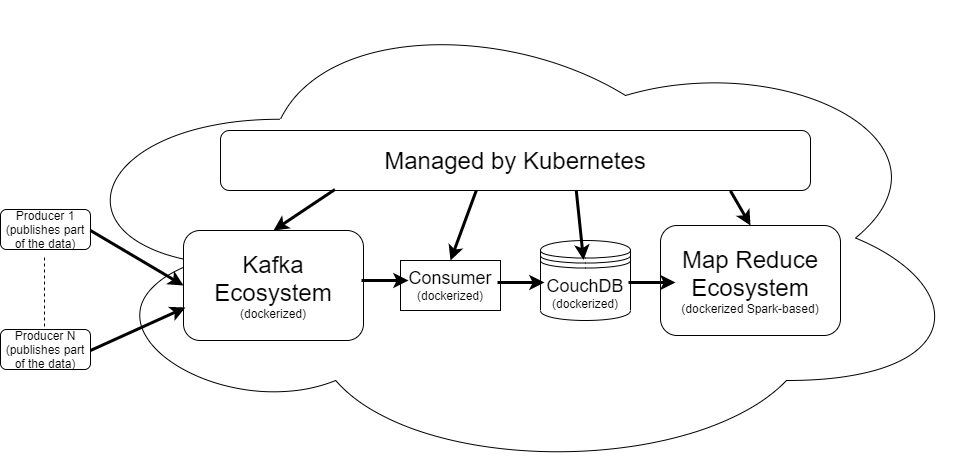
**CS4287-5287: Principles of Cloud Computing**

**Programming Assignment #4: MapReduce + Docker (Emulating IoT Analytics)**

Handed out 04/15/2021; Due 04/30/2021 (11:59 CDT)

**Please do not wait till the last minute to work on the assignment as you will need to develop map reduce code in Spark and modify your producer and consumer slightly.**

In this assignment we will extend Assignment 3 functionality with MapReduce capability. The assignment will enable us to apply our understanding of MapReduce and K8s-managed Docker containers to solve an IoT data analytics problem in the cloud using Apache Spark. To that end, we will build upon our Assignment 3 and expertise running a MapReduce program, such as the WordCount, and apply it for the energy data set example (explanation below). The overall setup of your assignment will be as shown in the figure below.



**Figure 1: IoT Analytics Workflow**

We will use Apache Spark to solve the energy example. The energy application is taken from the ACM DEBS 2014 Grand Challenge on Smart Homes described here: <https://debs.org/grand-challenges/2014/> and snippets of which are reproduced below.

*The ACM DEBS 2014 Grand Challenge is the fourth in a series of challenges which seek to provide a common ground and evaluation criteria for a competition aimed at both research and industrial event-based systems. The goal of the Grand Challenge competition is to implement a solution to a problem provided by the Grand Challenge organizers. The DEBS Grand Challenge series provides problems which are relevant to the industry at large. DEBS Grand Challenge problems allow for evaluation of event-based systems using real-life data and queries.*

***Data***

*The Grand Challenge 2014 data set is based on recordings originating from smart plugs, which are deployed in private households. A smart plug plays a role of a proxy between the wall power outlet and the device connected to it. It is equipped with a range of sensors which measure different, power consumption related, values. For the purpose of the DEBS 2014 Grand Challenge, a number of smart plugs have been deployed in private households with data being collected roughly every second for each sensor in each smart plug. It has to be noted that the data set is collected in an uncontrolled, real-world environment, which implies the possibility of malformed data as well as missing measurements.*

*For the DEBS 2014 Grand Challenge we assume a hierarchical structure with a house, identified by a unique house id, being the topmost entity. Every house contains one or more households, identified by a unique household id (within a house). Every household contains one or more smart plugs, each identified by a unique plug id (within a household). Every smart plug contains exactly two sensors:  
(1) load sensor measuring current load with Watt as unit (2) work sensor measuring total accumulated work since the start (or reset) of the sensor with kWh as unit.*

*The schema of the base stream is following:*

***id****– a unique identifier of the measurement [32 bit unsigned integer value]****timestamp****– timestamp of measurement (number of seconds since January 1, 1970, 00:00:00 GMT) [32 bit unsigned integer value]****value****– the measurement [32 bit floating point]****property****– type of the measurement: 0 for work or 1 for load [boolean]****plug\_id****– a unique identifier (within a household) of the smart plug [32 bit unsigned integer value]****household\_id****– a unique identifier of a household (within a house) where the plug is located [32 bit unsigned integer value]****house\_id****– a unique identifier of a house where the household with the plug is located [32 bit unsigned integer value]*

**What to accomplish?**

The goal is to compute the average work performed and average load imposed on each unique smart plug, where the uniqueness of a plug is defined by the tuple <id of the house, the household and the plug> taken together. So, the key used for the map reduce process is this 3-tuple.

**How to proceed?**

* You can reuse your entire Assignment 3 setup (don’t take it down; reuse). Then use your additional code to run the map and reduce jobs in containers (inside pods). Use Kubernetes to deploy the map and reduce workers using its built-in load balancing strategies. One of the pods will be the designated the Master pod for the Spark Map Reduce framework. Have the final results available in one place, maybe even in CouchDB. Thus, the result will be a record <key, average work, average load> where the key itself is that 3-tuple.
* Using the data file that is supplied, use your producer code on your laptop to mimic the role of IoT sensors. Since you are a team of two, you could split the data file into two and have each producer stream half the contents. In other words, the producers then split the given files and stream the data (say 1000 JSONified records at a time; do not send one record at a time as it will be extremely slow) via Kafka and consumer into the CouchDB. The MapReduce master then reads from CouchDB and spawns the desired number of map and reduce tasks. Try for different values of M and R, say the following combinations: M=10, R=2; M=50, R=5; and M=100, R=10. Time the experiment start to finish. Run each for some iterations (at least 10) to get more realistic response times and 90th/95th/99th percentile values and plot them.

Please use your previously used floating IP addresses on Chameleon.

**Submission:**

Create a zip file with your code and submit one per team. Demonstrate the working of your code to the TAs.

**Spark background**

See DIY\_SelfStudy folder for Week8 slides on Apache Spark (should have been Week 12), and lecture recording on 4/15/2021.

**Milestones**

***Milestone 1***

* Keep your entire setup from Assignment 3 as is.
* Split the huge data file (provided in the Assignment 4 directory) into two parts so that each producer can read half the data.
* Modify your producer code so it can read half the data, and takes, say 1000 records at a time, and sends them to Kafka
* Kafka should then stream it to your consumer
* Modify the consumer such that it can now take these incoming records and dump them into CouchDB

***Milestone 2***

* Deploy Spark (you can deploy this manually; no Ansible; Docker and YAML files will be supplied)
* Develop appropriate code using Spark programming API to solve the MapReduce problem
* Develop some client code which will take the input data from CouchDB and sends it to Spark and then Spark dumps the results back into CouchDB as a different database.