**Learning Distributed Systems with Docker and Actor Model**

**Note**, create an account and a private project repo on gitlab.com. You can add me and a grader as observer (gitlab accounts: **peiworld**, **sp3673**). Grade will be assigned based on the **git commit** and **push** records. In addition, please have a read the following article and learn about how to publish your docker image.

https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04

Myself and a grader will download your Docker image to verify your project outcomes.

**Project 1: Learn about Docker, Chord, and Publish/Subscribe system**

*Objectives: This project helps students to get familiar with recent advance in virtualization technologies, in this case Docker. Also, students will learn and prototype a Pub/Sub system and Chord DHT algorithm.*

* Install and learn about Docker (docker.com). Please use **docker pull ubuntu** to download the image.
* You can either run docker images within your computers or VMs provided by the department
  + $> ssh USERNAME@docker00~07.cs.rit.edu

1.1: Implement Publish/Subscribe system (10%)

* Extend the provided classes for modeling a pub/sub system, topics, and events.
* Create appropriate data structure for storing topics, events, and user info
* Implement the five key operations (subscribe, unsubscribe, publish, advertise, notify)
* Implement functions to display the current list of subscribers on the event manager (need multi-threading in terminal)
* Provide necessary command line interface to support key operations; for example, describing event of interest
* Support asynchronous event notification: when a subscriber is offline, the system caches events intended for this subscriber and send to the subscriber when reconnected.

1.2: Implement Chord distributed hash table (10%)

* Model elements that you will need in developing Chord, for example, object classes for providing structured format for data object.
* Create a network of 16 nodes (docker containers)
* Implement a Chord DHT using these nodes
* Write code to split up the sample dataset into chunks (100 numbers each) and distribute them to the DHT
* Write code address fault-tolerance when nodes join and leave (moving data)
* Demonstrate the ability to perform query of data chunks;

**Project 2: Learn the Calvin framework and develop distributed algorithms (20%)**

*Objectives: This project is designed to provide hand-on experience with modern distributed systems. Students will learn about the traditional data processing in chunks (reflecting the map&reduce model).*

* Learn about the Calvin framework (search EricssonResearch/calvin-base on Github).
* Study the *minitutorial* and other documentations within the wiki page. Especially, you **MUST** study these MS theses:
  + [Mobile Devices In the Distributed IoT Platform Calvin](https://lup.lub.lu.se/student-papers/search/publication/8912745)
  + [Dynamic Fault Tolerance and Task Scheduling in Distributed Systems](https://lup.lub.lu.se/student-papers/search/publication/8876351)
* Use the same docker images: **docker pull peiworld/calvin-iotx:latest** to run multiple Calvin runtimes
* Explore and run the demo applications within calvin-base/calvin/examples
* Develop your own actors and let them talk to each other across different runtimes
* Migrate Calvin actors from one runtime to another
* Implement **three** actors that read file, compute gradients, and print result
* Follow this GradientDescent example <https://github.com/peiworld/GradientDescentExample> to develop your linear regression algorithm. You are allowed to modify and improve the algorithm to speed up the computation. Note, the goal of this task is to find the values of **m** and **b** in a function **Y=mX+b**. You are given three sample datasets (each line is **X,Y** values) to start with, but you might want to use my example scripts to generate datasets of your own.
* Distribute these actors on three separate Calvin runtimes
* Parallelize the computations into **five** pipelines; this requires you to modify the **readfile** actor to perform byte-range read, and your **printresult** actor to merge the results from different models (e.g., by averaging the resulting m and b values).

**Example dataset with 100 random data points and performance time**

Basic statistics of this sample dataset

N=100 # 100 points

m=1.234 b=432.1

# rando seed for offset the value of y

r\_start=-100 r\_end=100

100_samples.pdf

mac:GradientDescentExample $ time python gradient\_descent\_example.py myDataset/100\_samples.txt 10000

Starting gradient descent at b = 0, m = 0, error = 192125.797919

Running...

After 10000 iterations b = 78.1446964147, m = 0.542199793418, error = 130348.959925

real 0m2.573s

user 0m2.299s

sys 0m0.112s