CPSC454-01 Phase 3 Progress Report

Project Members:

Christian Angeles:

- Email: christian.angeles@csu.fullerton.edu
- Major: Computer Science (Undergraduate)

John Zavala:

- Email: johnzy27@yahoo.com
- Major: Computer Science (Undergraduate)

Project Title: Distributed Cloud Computing with StarCluster (DC2S)

Topics:

- Distributed Computing
- Mobile Cloud Computing
    - Android platform; computational offloading to cloud server

Project Description:

Our goal is to simulate computational offloading of a compute-intensive task on an android application to a cloud server. This server will complete the task via distributed computation. The data will be an array of at least 1,000,000 elements conceptualizing a large data set. Data that is too large to store on an android device. The program on the cloud server will be performing an exhaustive search and comparison on such data to simulate the computational offloading. In order to create a compute-intensive environment, the algorithm is going to search and compare the 1,000,000-element-sized array with itself. Forcing the algorithm to perform with a time complexity of $O(n^2)$. A slower run time will allow us to see a difference in computational performance.

Project Setup Environment:

Amazon Web Services (AWS) EC2, StarCluster, Message Passing Interface (MPI), Android

Project Skills Needed:

Languages: Java, Python

Operating System: Android, Ubuntu

Completed Task List:

<u>Amazon Web Services (AWS) EC2</u>

- Account creation and familiarization
    - Created instances

        - Security group configuration
        - SSH connections to AMIs using private keys
- Implemented cloud server environment with StarCluster toolkit

<u>StarCluster</u>

- Learned how to use the toolkit to manage cluster (terminal commands)
    - start, stop, terminate, and create a cluster
    - upload/download files to nodes
    - adding/removing a node
- Configured settings to automate creation of cloud cluster
    - Ubuntu 13.04 starcluster AMIs (master and two slave nodes)
        - Updated to 4 nodes (master and 3 slave nodes)
    - Network File System (NFS) sharing with Amazon's Elastic Block Storage (EBS)
    - Security Group (inbound/outbound SSH connections)
    - Password-less SSH by creating a public keys for VMs

<u>Message Passing Interface (MPI)</u>

- Learned how to implement a program in C/C++ and Python
    - mpi4py library for Python
    - Initialize and Finalize MPI program
    - rank/process
    - size of the MPI "world"
    - communication between processes
        - Point-to-Point Communication
        - Collective Communication
            - Broadcasting
            - Scattering and Gathering

<u>Cloud Server Program</u>

- Written in Python
    - Simpler to write MPI program in Python
    - Saves time from compiling C/C++ code
    - MPI Initialize and Finalize is done for you
- MPI program
    - Divides the array into segments
        - Number of segments are based on number of jobs are allocated to complete the task
            - Programmed to dynamically scale
    - Array elements are initialized to 0 except for the last index
    - The algorithm to search and compare for a number greater than 0 is a simple double for-loop
        - Number is placed at the last index for the worse-case time complexity
    - Implemented randomization
        - Random index will be assigned a number greater than 0
        - Shows dynamic scaling and expected run time
    - Implemented display output
        - Identify which process belongs to which node
        - Implemented timer
            - Shows how long it takes for the program to finish
    - Program optimization
        - Increased array size to 1 million from 100,000
        - Considered cloud resources for multiple client connections
            - Reduce to 1 process for each node per client connection
            - Implemented multithreading to process data
    - Implemented message queue for main process and MPI processes
        - Communication between main process and MPI processes
            - Main process manages socket communications
        - Reused random index number as a "cookie" value for each client connection
    - Implemented multithreading for MPI process
        - Each process on the cluster creates a processing thread

            - Process will work on a segment and create threads for other segments
        - Master node process divides array and allocates segments to other process
            - Data thread created to divide array into segments
                - Used MPI point-to-point communication to send segments to other nodes

<u>Socket Programming</u>

- Implemented socket communication between Android application and cloud server
    - Cloud server listens for client connections on a thread
    - Creates additional thread for each client connection
    - Thread allows for blocking code execution
        - Main program can focus on primary task
    - When simulation completes, server sends completion time to Android application
    - Similar concept above applied to application socket programming

<u>Android Application</u>

- Main UI thread
    - Simple user interface design
        - Implement button to start simulation
        - Displays cloud server completion time
- Multithreading for socket connection
    - Used existing socket API for Android Studio
        - AsyncTask
            - Implemented socket connection to cloud server
            - Requests cloud server for task migration simulation
                - Waits for a response and closes socket connection
                - Interprets data from cloud server