Bosley's Distributed Computing System

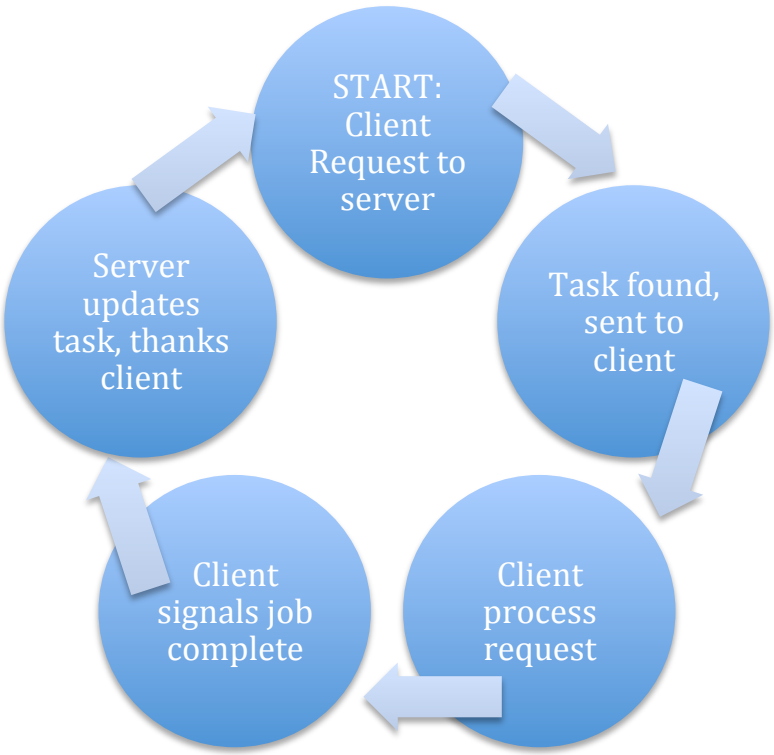
Josh Bosley
CSCI 434
Final Project

As computers advance, so too do our computational needs. In order to meet these needs it is imperative that we come up with techniques to maximize the speed of which we calculate problems. One way we could do this is through advances in computer hardware. Advances in hardware potentially allows us to crunch more numbers in less time, but the cost of replacing existing hardware is not always financially viable. A more suitable solution for faster calculations in many cases is found through changing how we think about the problems at hand within software. Changing the way we approach the actual calculation of the problems at hand can yield fantastic results. In computing the total run time of a set of problems can be computed in XT/K time, where X is the number of problems to compute within the set, T is the amount of time it takes to calculate each problem, and K is the numbers of computers the tasks are being distributed to.

This means that if we were given 5 problems that took 2 minutes each to solve, conventional means (K = 1) would take 10 minutes to solve the whole set. One way of maximizing computational speeds is a technique known as Distributed Computing. Using this method of computing, we aim at increasing K from 1 to N where N is any positive real number greater than 1. If we were to set K = 5 in the above example, the total amount of computation time would then be 2 minutes.

My final project demonstrates the distributed computing technique using a QT C++ client, a Multi-Threaded Python 2.7 socket server, as well as a MySQL database. With my publicly available server h9k.lssu.edu hosting the python socket

server, my distributed computing system (DCS) can be accessed by multiple clients simultaneously despite geographical location. The DCS has been created to be modular in terms of what the client can process, as-to allow a DCS framework-of-sorts for easy expansion. Keeping modularity in mind, I also created the server to be easily expanded upon in that the only thing required for functionality to be added is the function call for sever-side data retrieval to an expected client-code within the *request_map.py* file. In its current state, my DCS is fully operational and currently functions as a run-time analyzer for sorting functions on randomly generated data. Once the data from the server has been run a set amount of times against the sorting algorithms, the run-time information is reported back to the server and stored into the MySQL database. While what is being computed (sorting) is not particularly demanding, the underlying concepts of distributed computing are laid out in full.



The cycle above depicts the generalized DCS logic flow, starting from the top-most node. The client will signal to the server that it would like a task to complete. Once the server receives this request, it selects the next task within the database and sends it to the client. The data sent to the client contains the operational data, in addition to information as it relates to the data so the client can understand how to operate on it. At this point, the server will move the sent task to an "in-process" table to ensure no other client receives the same task then closes the connection. Once the client has the data from the server it determines what specifically must be done and initiates the process until completed. Upon completion, the client will send the report information to the server along-with the task information so the server can move the job from the "in-process" table to the "completed-table" which will contain the data, in-addition to the task information generated by the client. The client will wait for update confirmation prior to being allowed to submit another request to ensure the safe transfer of task a task from "in-process" to "completed." In the event that a task is left in the "in-process" table for a certain period of time it can be triggered to be moved back into the "open-tasks" table.

When it comes to completing a set of problems in a smaller amount of time, my DCS can essentially set the K value mentioned above up-to the maximum load of the server that hosts the DCS socket server. Coupled with the modularity of the client/server applications themselves my DCS can be easily set to distribute and compute any number of different problems. If configured to do so, my DCS can cut the total time spent on calculations to a fractional amount of time that it would take for a single machine to compute.