CS 423 MP4: Report

Chengyin Liu (liu189), Lin-Ming Hsu (lhsu7), Rohan Sharma (sharma27)

December 2 2011

Our object structure mirrors the components given in the flowchart, except for that we directly use the hardware monitor for worker thread running.

For our workload, we decided to choose matrix-matrix addition, with each job being addition of two integer elements. Actually, we add the second matrix 1000 times to observe the load balancing fully. While this choice of jobs might not be the most efficient design decision, note that load-balancing, and not the workload itself, was our main focus. In fact, the workload is abstracted away by means of generic superclasses.

For networking, we use TCP/IP for convenience and efficiency, and we adopt a client/server architecture for remote and local nodes; also, we have separate runners for each, though the overall design for both are similar. Our transfer and state channels correspond to two different ports. The transfer and state managers are similar in general structure: we use *server-initiated state and job transfers* to avoid synchronization complications.

We run the state manager every tenth of a second. The state manager dispatches a call to the adapter during every sending of state. We have a state handler that listens to the sending of state.

The transfer mananger sends jobs in bulk. In case the queue is empty, it singals to the client by sending Dummy object instead of a Job.

For load-balancing, we've just considered the CPU utilization and throttling. We take the balance in CPU usage weighted with the number of jobs in the queue. We then send/receive jobs in proportion to the balance.

We simply use a Java library thread-safe data structure for our job queue. This keeps synchronization between different components simple while making it as multi-threaded as possible.