Operating System Design MP4 Report

Pin-Xuan Lee (plee18)
Yu-Lin Chien (ychien8)
Shuo-Yang Wang (swang234)
Wei-Tze Tsai (wtsai10)

Contain files:

my_mp4.py GUIServer.py ping.py run_local.sh run_remote.sh GUI/gulpfile.coffee GUI/package.json GUI/src/coffee/index.coffee GUI/src/jade/index.jade GUI/src/sass/index.sass

Implementation and design decisions:

In MP4, we use python for implementation: my_mp4.py. The local and remote virtual machines share the same code with only different input parameters.

Local: `python my_mp4.py local <throttling>`

Remote: `python my_mp4.py remote <throttling>`

Class MP4Job:
An MP4Job contains 3 elements: ID, startPoint, and A. ID is the job index, startPoint stores the starting index from the original array, and A is the data array which will keep the value of each entry of the original array.

toObj(): Pack the 3 elements as an object for the further usage.

Class ServerHandler:
A ServerHandler handles an http request. Here we use http post for all the message transfer.

do_GET():

do_POST(): process the received message according to its type: *job, signal, result,* and *state*.

Class TransferManager:
A class that manages job transfer.

transferJobs(): transfer a MP4Job object to the other virtual machine.

Class StateManager:
A class that manages state transfer. It contains 4 variables, *partnerJobNum*, *partnerThrottling*, *delay*, and *bandwidth*. *partnerJobNum* and *partnerThrottling* store the other machine's number of jobs and throttling value, while *delay* and *bandwidth* store the current machine's delay and bandwidth.

setPartnerVariables(): Update the states of the other machine once a *state* message is received.
stateToObj(): Similar to toObj(). Pack all elements as an object for the further usage.
startStateSendingThread(): creates a thread specifically for sending states (calling sendState()) throughout the process.
sendState(): use HTTP POST for sending states to the other machine. STATETRANSFERPERIOD is the parameter that determines the period of state sending. It is set to be 0.5.

Class HardwareMonitor:
A class that catches the CPU usage from the hardware. It contains one variable *CPUUtilization* for recording the CPU usage.

startHardwareThread(): Create a thread specifically for updating the CPU usage (call updateCPUUtilization()) throughout the process.
updateCPUUtilization(): Fetch the CPU usage in percentage for every 2 seconds.

Class Adaptor:
A class makes the decision whether a job should be transferred to the other machine according to all gathered information. It has a variable isAdaptorTransferThreadStart.

startAdaptorTransferThread(): Create a thread specifically for transferring jobs (call _transferJobIfNeeded()) throughout the process.

_transferJobIfNeeded(): If shouldTransfer() returns true, then transfer a job by calling transferManager.transferJobs().
shouldTransfer(): Implement the transfer policy. We implemented in sender initiated version. Send a job to the other machine if the number of job of the machine is greater than the other machine's number of job plus one. Therefore, we get rid of the chance of oscillation.

Global Variables:
jobList: A list keeping the unfinished jobs.
resultList: A list keeping the results of the finished jobs.
throttling: A variable given by user which determines the working time in a period.
isLocal: A variable records whether itself is the local or remote machine.
isRemoteTerminate: A variable records whether the remote machine terminates.

Global functions:
job_decoder(): A helper function for json decoder.
startListenThread(): Create a thread specifically for listening to http requests throughout the process.
transferResults(): Transfer *result* to local machine.
getJobSeparationArray(): Determine the starting element index of a job.
_initJobToJobList(): Initialize jobList.
_bootStrapTransferHalfJobs(): Transfer half of jobs to the remote machine.
aggregationTransferResults(): Transfer results back to the local machine.
doJob(): do jobs and update resultList.

Main:
BOOTSTRAP
1. Start listen, state sending and hardware threads.
2. For the local machine, send half of the jobs to the remote machine.
EXECUTION
1. Start adaptor transfer thread.
2. Do jobs under the throttling setting.

AGGREGATION
Gather results on local machine.
TERMINATION

Terminate the process.

Bonus:
zlib.compress() and zlib.decompress():

For compressing and decompressing the jobs. We chose to compress our job before sending it because there is a huge difference between with compression and without compression.

We conducted an experiment just transferring 100 jobs from local to remote machine. It takes 10.9945600033 seconds without compression, and 1.08475399017 seconds with compression. The huge difference makes us to determine to compress data before the transmission.

ping.py:

Get network bandwidth and delay.

GUI:

Web based GUI. We launch up a server (GUIServer.py) for our webpage to solve the cross-domain issue.