Name	NetID
Juanli Shen	jshen35
Funing Xu	fxu12
Wei Yang	weiyang4
Yunsheng Wei	wei29

1. Usage
   - first start remote node:
     python remote_node.py
   - then start local node:
     python local_node.py
   - command line interface for getting and setting CPU throttling value (case-sensitive):
     get
     set X (0 < X <= 1.0)

   - if you want to modify configuration parameters, look into:
     transfer_policy.py & constant.py

2. Design & Implementation
   2.1. State information policy
   State manager periodically sends its local state to peer node.
   2.2. Transfer policy
   Every node periodically checks whether a load transfer should occur given its local state and peer's state. (we only use job queue length and CPU throttling value). We calculate estimated completion time as job_queue_length / cpu_throttling_value, and all transfer decisions are based on local's estimated completion time and peer's estimated completion time.
   2.2.1. Sender-initiated transfer
   If my estimated completion time - peer's estimated completion time > a threshold, then transfer a part of my job to peer to balance our estimated completion time.
   2.2.2. Receiver-initiated transfer
   If peer's estimated completion time - my estimated completion time > a threshold, then transfer a part of peer's job to me to balance our estimated completion time.
   2.2.3. Symmetric-initiated transfer
   The hybrid of sender-initiated transfer and receiver-initiated transfer.
   2.3. Selection policy
   Choose the first job in the job queue to transfer.
   2.4. Location policy
   Trivial.

3. Experimental Evaluation
   We test the performance among different transfer policies. But actually there's minimal difference. The job finish time is 64s, 63s, 63s. The transferred job size is 88, 111, 97. The

transferred data size is proportional to the job size. This is reasonable, because our implementation means to achieve that.