Report for PA3

Jian Zhu
A20362556

1. Design

1.Local

I develop this program on Eclipse in Java language. There are two classes in my program. One is client, the other one is worker. At first, in the client class, I read the command 'sleep 0' , 'sleep 10', etc. from txt file and store this command with a ID (from 0 to the total number-1) in String format in a queue, called 'LinkedBlocakingDeque', list. Then the worker class read the string format command from the queue and execute this command with Thread.sleep("sleep command"), if success, return a successful message to the other queue called 'LinkedBlocakingDeque', outlist. In the client, it will show how many tasks succeed and what they are.

Also, I will do the same operation varying level of concurrency(1,2,4,8&16 threads). The executable class is in main.java with a main function to decide how many threads to start. I also measure the execute time from the first tasks the client put in the queue to the last tasks executed by the worker.

Since LinkedBlocakingDeque is a Deque which will block if a thread attempts to takes elements out of it while it is empty or the elements is being taken by other thread. So different threads will not execute the same command in the queue.

2.Remote

For the remote version, I also develop it on Eclipse with the amazon API. At first, I create two SQS queues in the amazon website, also create dynamoDB table through the code, the table has a primary key named 'id', another is 'status' to show the whether this item is used, if used, value turns to '1', if not, stays '0'. Then executes the worker class to check the first SQS queue, if empty, keep waiting until there are messages on the first SQS queue. Next, execute the client class to put the tasks, which is read from txt file, in the SQS queue, also, store each item in the DynamoDB with ID and status (0). Now the worker class

can read the task from the first SQS queue, then check on the DynamoDB table to see whether it is executed with the status value, if 0 executes them,(sleep workload), then delete this task from the first SQS queue; if 1, skip. After that (successful execution), store the successful messages in the second SQS queue. At the same time, the client will receive messages from the second SQS queue to see how many tasks are executed successful and what they are.

Also, I will do the same operation varying the number of instances(1,2,4,8&16). Then measure the execute time from the first tasks the client put in the queue to the last tasks executed by the worker.

The function of DynamoDB is to check whether the current task is being used by another worker in case of duplicate execution.

2. Manual

1. Local

For the local type, there are total 3 directories, src, bin and task. For src and bin, each contain 3 files, client, worker and main. For task directory, there are 4 txt files, each execute sleep 0, sleep 10 sleep 1000 and sleep 10000. In the bin directory, the main.class is the executable file, run this file in the terminal, like
'java thread_number task_number file_address'.
The thread-number should be 1,2,4,8 or 16.
The java running environment is java1-8.

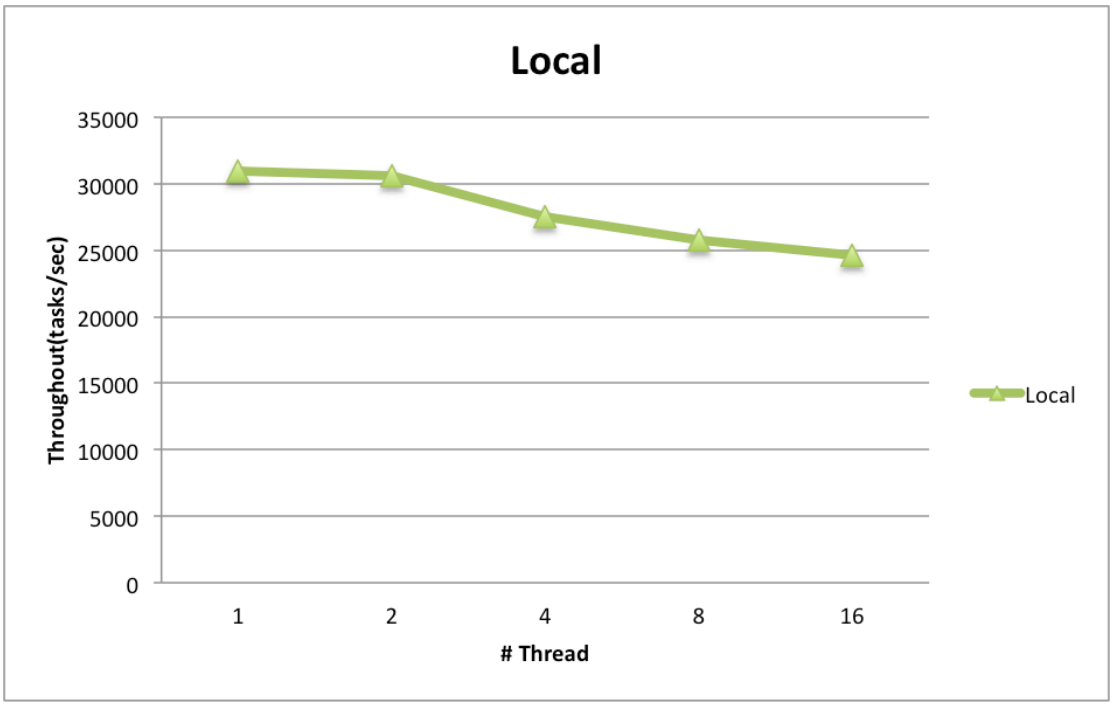
2. Remote

For the remote version, there are two directories: bin and src, and .jar file. First, in the terminal, run the AmazonDynamoDB.class like 'java AmazonDynamoDB' to create a DynamoDB table, also need to create two SQS queues before start. In the terminal, run remote.jar file using the following command order:
'java -classpath remote.jar testproj2.worker' to run worker.
'java -classpath remote.jar testproj2.client' to run client.
In order to evaluate the performance in different size of workers on AWS, you can open multiple terminal windows to run the first command.
The java running environment is java1-8.

3. Performance Evaluation

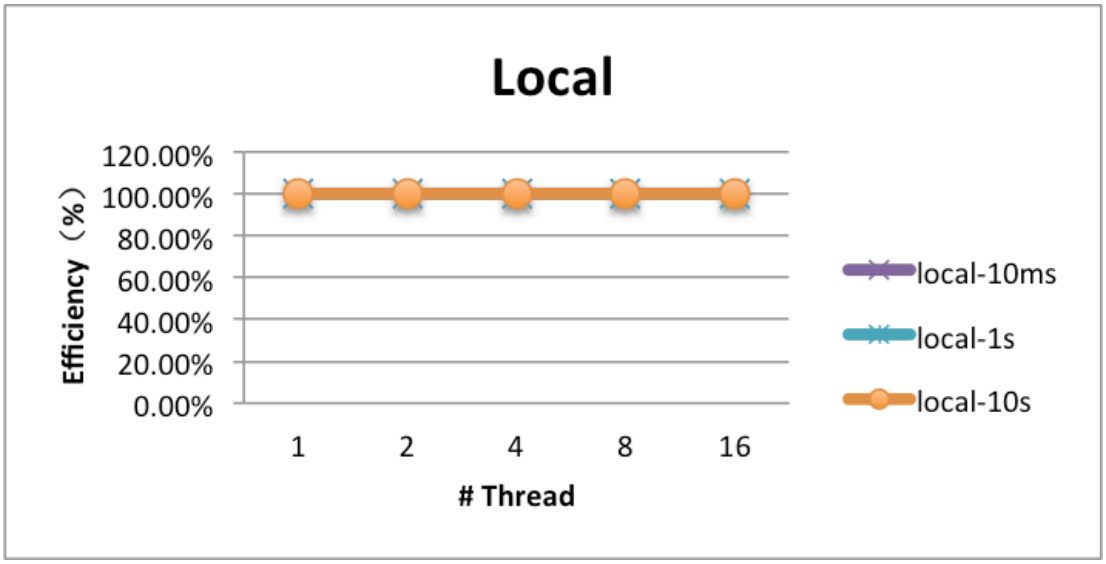
1. **Local**

Throughout:



From the figure above, we can see that with the number of thread increases, the throughout decreases. That is because there is extra cost of synchronization between multiple threads.

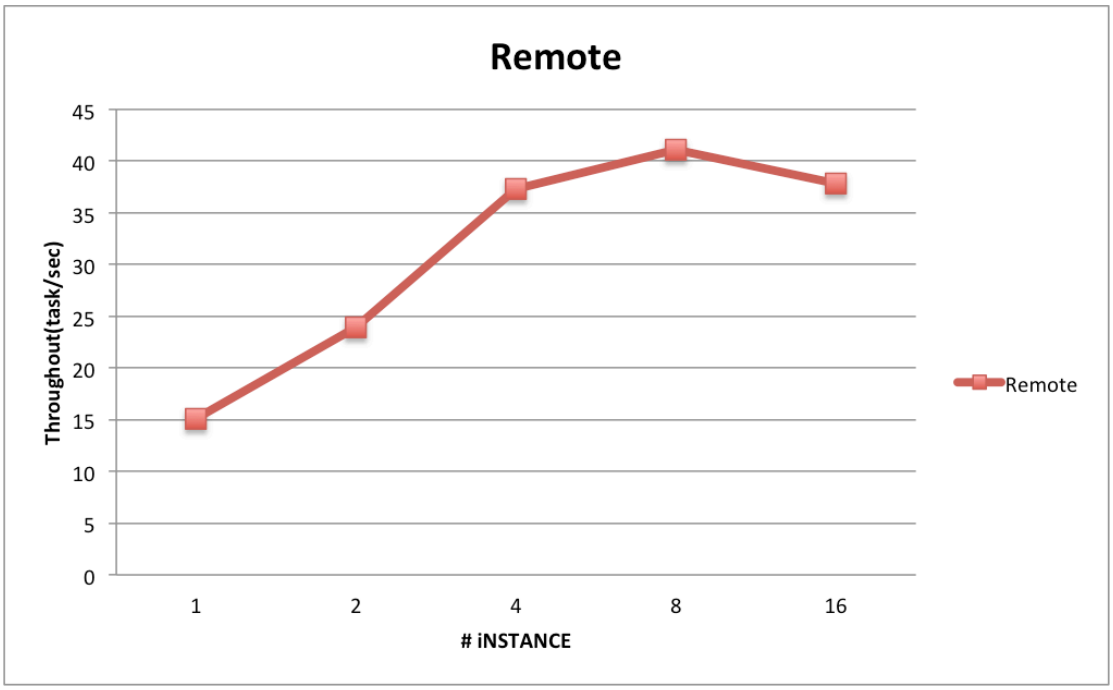
Efficiency:



From the figure, we can see that the efficiency with different workloads in different threads keeps stable, nearly 100%. That is because there is no much network communication during different thread and also

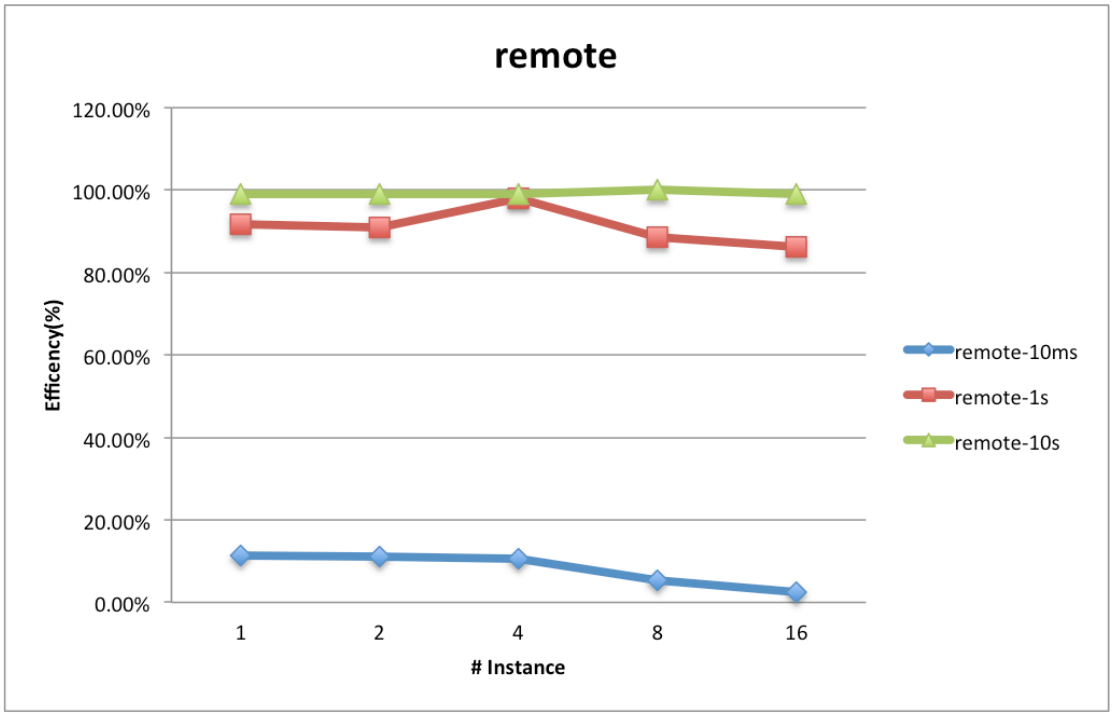
2. **Remote**

Throughout:



From the figure above, we can see that with the number of instance increases, the throughout will not always increase. Before the #instance scale to 8, the throughout will increase, but when the # instance scale to 16, the throughout decreases. The result may be due to the bottleneck of client. There are extra cost of network commuicatin on multiple instance.

Efficiency:



As the figure shows, the efficiency is stable when running the same tasks with the number of instance increases, only a little decrease is because that the TCP connection on too many instance. But the efficiency drops for shorter tasks(workload from 10s to 10ms). That is because that there are two many workers for task distribution, the scheduler cannot have a load balance and some workers remain idle.