Amna Irfan
CS 441
09/30/19

Modeling, Simulating and Evaluating CloudSim Toolkit

Introduction: Cloud Computing has become an integral part of businesses that offer software as a service. The cloud computing model promises these companies unlimited, scalable and elastic computing resources along with the promise of high availability, high reliability and robust fault-tolerance mechanisms as well as a reliable and secure network connection. However, these promises put a great amount of responsibilities on the shoulders of cloud computing providers. One might think that data centers are only responsible for housing commodity computers and configuring the computers' software and hardware in a manner that allows customers to remotely use them. However, this is far from the reality. Cloud infrastructure providers have to provide services to thousands of customers mostly at the same time which means they need to come up with strategic policies for resource management, resource allocation and job scheduling. Since their own business model depends on that, their policies need to be cost and time efficient.

Cloudsim is a framework that allows developers to model and simulate cloud computing infrastructure and services. Developers can fully customize the default model using some of the following parameters;

1. Users can add multiple datacenters and connect them using different types of network topologies.
2. Each datacenter can host multiple physical hosts. Users can add more hosts to make the datacenter more powerful.
3. Alternatively, users can increase the number of processing units on each host.
4. The user can increase/decrease ram, storage capacity and bandwidth of the host.
5. The user can do 2-4 for virtual machines as well. VM are submitted to the data center broker and then assigned to actual hosts in the datacenter based on the VM scheduling policy.
6. The user can update the VM Allocation policy, VM scheduling policy and cloudlet scheduling policy.

In my simulation, I worked with cloudlet scheduling policies. These policies allows virtual machines to decide the order in which cloudlets will be executed. I ran simulations on the two cloudlet schedulers from Cloud sim (Time shared, Space shared) and I created four of my own (FIFO, LIFO, SJB, Priority) custom scheduling policies. These four custom scheduler classes were extended from the space shared scheduler.

1. Time Shared Cloudlet Scheduler: In this policy, instead of queuing and executing cloudlets in batches on the virtual machine, all cloudlets are executed in parallel. This ensures that one cloudlet doesn't get priority over another but this increases the average execution time of the cluster.
2. Space Shared Cloudlet Scheduler: In this policy, each core is being used by a single cloudlet. Once the cloudlet completes, the core is assigned to the next cloudlet in queue.

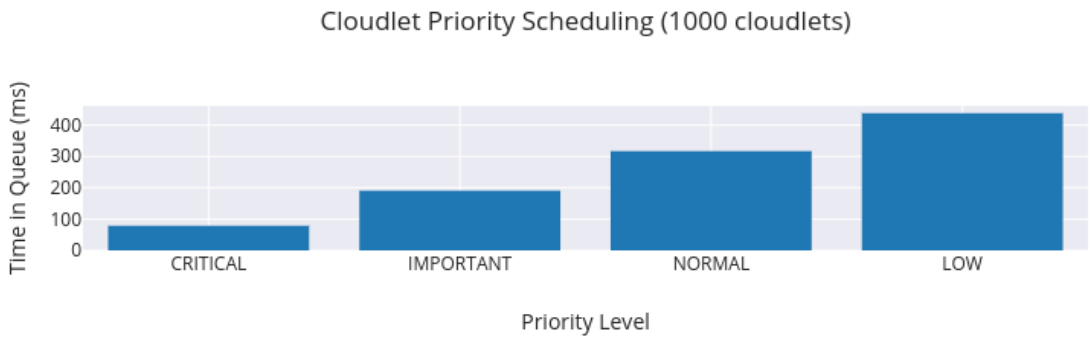
3. First in First out Cloudlet Scheduler: This turns out to be the same as space shared as space shared implements a FIFO queue.
4. Last in Last out Cloudlet Scheduler: This policy is the opposite of FIFO. Even though for some jobs a LIFO queue is optimal, that is not the case for cloudlet scheduling.
5. Shortest Job First Cloudlet Scheduler: This policy queues cloudlets based on the length of the cloudlet. The cloudlet requiring the least resources is executed before the ones requiring more CPU time.
6. Priority Cloudlet Scheduler: For this simulation, I extended the cloudlet class to store it's priority level. During creation, each cloudlet is randomly assigned a priority by the data center broker (just for this simulation purpose). The priority levels are
    a. Critical
    b. Important
    c. Normal
    d. Low

   The cloudlets are then executed in order of priority. For example, a cloudlet with critical priority is executed before the one with low priority.

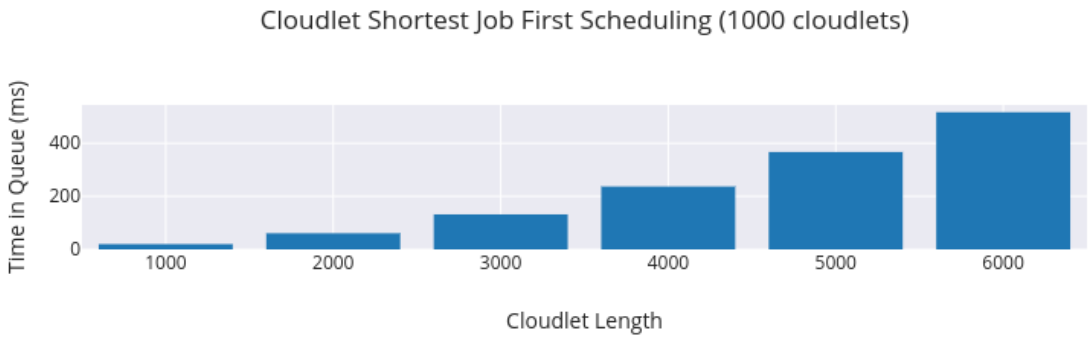
   One thing to notice here is that the cloudlet scheduler policy isn't used for assigning cloudlets to virtual machines. The cloudlet policy is only used after cloudlets have been assigned to a virtual machine. This means that even if we use a priority cloudlet scheduling policy, it is possible that cloudlets that with a low priority finish before the ones with a higher priority as it could be that the low priority cloudlet got assigned to a virtual machine hosting cloudlets with a similar priority.

Results:

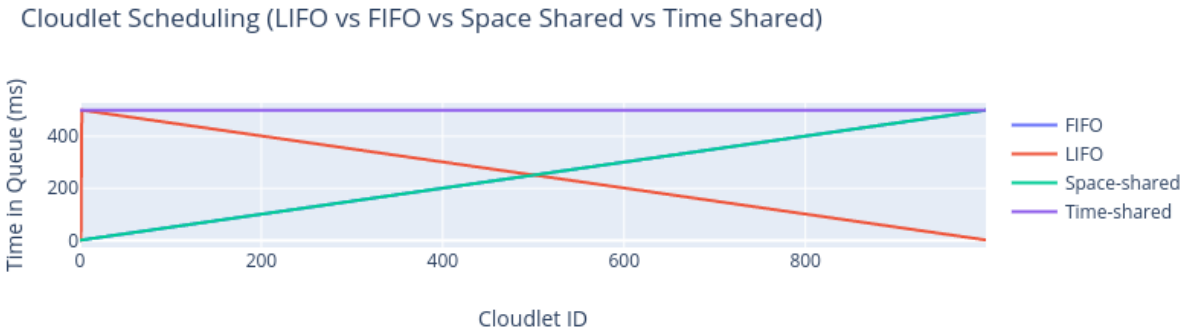
1. We used the same configuration for all six simulations. The only difference between them was their cloudlet scheduling policy. However, for the shortest job first simulation, we generated a random length for each cloudlet (1000-9000).
2. In the chart below, we see that after implementing the priority queue, "critical" cloudlets on average were the first ones to leave the queue followed by the "important" cloudlets.



3. Again, for SFJ (shortest job first) the jobs with the shortest length left the queue the fastest. Another reason we see a greater difference between them is because short length cloudlets in general take lesser time.



4. In the chart below, we see FIFO scheduling policy works the same way as space-shared policy (both lines are overlapping). We also see how both FIFO and LIFO are opposite in the graph. Also note, that due to time shared scheduling policy not implementing a queue and running cloudlets in parallel, all cloudlets stay in the VM for the same amount of time.



5. We see that the cost of using space-shared, FIFO, LIFO and priority scheduling policies is the same. The intuition behind them wasn't necessarily to reduce costs or CPU time, instead was to complete jobs in an order that is not arbitrary. Data Brokers can assign priorities to cloudlets based on the customer's history or even the cloudlet's history. Notice the we get a cost of $1.5M for time-shared, even though it shares the same configuration parameters as space-shared. This could be because we do not have enough resources in our datacenter to execute these many cloudlets in parallel. This increases the CPU time of all cloudlets which in turn increases the cost abnormally.
   Shortest Job First costs more than the other four because it has cloudlets of different lengths.

Cloudlet Scheduler Policy	Cost in USD (1000 cloudlets)
FIFO	$3,000
LIFO	$3,000
Priority	$3,000
SJF	$15,039
Space shared	$3,000
Time Shared	$1,500,000

   *Note: These can be verified by running the simulations in Scala.

Map Reduce: The simulated map reduce architecture includes the following four main components;

1. Map-Reduce App: This is the class the user uses to submit their map-reduce job to the master. In the call to submit map-reduce job, the user has to provide the following parameters
   a. Task Length
   b. A multiset of random key value pairs where both the key and value is a number.
   c. File Size
   d. Input Directory
   e. Output Directory
   f. Mapper Script
   g. Reducer Script
2. Master: This is the component that receives the map reduce job. The master takes on the role of a data-center broker. It spawns mapper and reducer cloudlet jobs based on the length of the task. In this simulation, we do not use pipelining. To simplify the job, the master waits for the mappers to complete before spawning reducers. The number of reducers are equal to the number of unique

keys the mapper finds. The master also sends messages (heartbeat) to both the mapper and reducer periodically to confirm they are still running. The master also calls another function periodically called “checkpoint” to save its state incase it shuts down.

3. Mapper: The mapper is an extended class of cloudsim’s cloudlet and is responsible for mapping the key-value pairs into a multiset of key-value pairs that are then sent to the reducer for the reduce operation. The mapper is also responsible for sending back a heartbeat to the master on request.
4. Reducer: The reducer is an extended class of cloudsim’s cloudlet and is responsible for reducing the multiset of key-value pairs found by the mapper. The reducer is also responsible for sending back a heartbeat to the master on request.

Sample Output:

*****Saved Master's State!!*****
********** Map/Reduce Input **********
(8, 4)
(5, 6)
(5, 5)
(4, 5)
(8, 7)
(4, 4)
(7, 7)
(7, 8)
(3, 4)
(3, 2)
(4, 4)
(6, 7)
(7, 5)
(2, 5)
(4, 7)
(6, 1)
(8, 8)
(8, 8)
(3, 8)
(5, 6)
(6, 1)
(6, 3)
(2, 5)
(6, 4)
(5, 2)
(2, 3)
(3, 5)
(6, 4)
(7, 5)

(7, 4)
(1, 8)
(4, 7)
(7, 8)
(1, 4)
(2, 2)
(8, 1)
(8, 4)
(1, 8)
(8, 2)
(8, 1)
********** Submitting Mapper Jobs **********
********** Output from Mapper 0 **********
{3=[4, 2], 4=[5, 4], 5=[6, 5], 7=[7, 8], 8=[4, 7]}
********** Output from Mapper 1 **********
{2=[5], 3=[8], 4=[4, 7], 5=[6], 6=[7, 1], 7=[5], 8=[8, 8]}
********** Output from Mapper 2 **********
{2=[5, 3], 3=[5], 5=[2], 6=[1, 3, 4, 4], 7=[5, 4]}
********** Output from Mapper 3 **********
{1=[8, 4, 8], 2=[2], 4=[7], 7=[8], 8=[1, 4, 2, 1]}
********** Final Mapper Output **********
{1=[8, 4, 8], 2=[5, 5, 3, 2], 3=[4, 2, 8, 5], 4=[5, 4, 4, 7, 7], 5=[6, 5, 6, 2], 6=[7, 1, 1, 3, 4, 4], 7=[7, 8, 5, 5, 4, 8], 8=[4, 7, 8, 8, 1, 4, 2, 1]}
*****Mapper ID: 0 running on VM 0 responded to heartbeat!*****
*****Mapper ID: 1 running on VM 0 responded to heartbeat!*****
*****Mapper ID: 2 running on VM 0 responded to heartbeat!*****
*****Mapper ID: 3 running on VM 0 responded to heartbeat!*****
********** Submitting Reducer Jobs **********
*****Reducer ID: 0 running on VM 1 responded to heartbeat!*****
*****Reducer ID: 1 running on VM 1 responded to heartbeat!*****
*****Reducer ID: 2 running on VM 1 responded to heartbeat!*****
*****Reducer ID: 3 running on VM 1 responded to heartbeat!*****
*****Reducer ID: 4 running on VM 1 responded to heartbeat!*****
*****Reducer ID: 5 running on VM 1 responded to heartbeat!*****
*****Reducer ID: 6 running on VM 1 responded to heartbeat!*****
*****Reducer ID: 7 running on VM 1 responded to heartbeat!*****
********** Final Reducer Output **********
{1=20, 2=15, 3=19, 4=27, 5=19, 6=20, 7=37, 8=35}

Process finished with exit code 0