ECE 459: Programming for Performance
Assignment 4

Your Name

April 4, 2018

Load Balancing Implementation

This assignment investigates the impact of the load balancing for a multi-server applications. The goal of a load balancer is to average out the load of the servers thus migrating jobs from over-loaded servers to underloaded ones.

The algorithm is implemented using a doubly linked list data structure to store pointers to the start and end of each server as well as the size of the server (in terms of amount of jobs currently queued). The load balancer is run proportionally to the rate of arrival (lambda) where its run 100x slower than the arrival rate. When the load balancer is called all the servers are locked to check the deviation in size from the average. The servers that have a size greater than the average by X amount have those X jobs removed into a temporary queue to redistribute to the serves that have X amount of the average missing thus balancing the load across the amount of servers.
Removing and adding of the jobs to queues becomes simple with a doubly linked list where there are access points from the start and end of the queues as well as having next and previous pointers keeping track of job order.

Results - Impact of Load Balancing

Disclosure: All tests are run on a 2016 (15 inch) MacBook Pro 2.9 GHz Intel Core i7 with 16 GB 2133 MHz LPDDR3 due to server overload on ecetesla0 and ecetesla00. Though the algorithm has been tested to run on the Tesla servers. Ted's generation method is utilized to see more of an effect on server overload.

The initial impact investigation is based on system parameters **P** outlined in Table 1.

P	Variation	Description
n	6	Number of servers (queues)
a	(1 2)	Assignment policy (1 for random, 2 for round robin)
j	2500	Number of jobs
b	(1 0)	Load balancing (0 for off, 1 for on)
l	(1000 100 10 1)	Lambda (arrival rate parameter)
m	(15000 12500 10000 7500 5000 2500)	Max Rounds (controls variability of job size)

Table 1: System parameter variation to test impact of the load balancing algorithm.

The results of 90th percentile average (of 3 runs per data point) response times for job competition are analyzed through Excel and data can be seen in Appendix 1 and 2. Data analysis is given the following section.

Discussion

Lambda Impact

From initial test data some trends are seen in the efficiency of the load balancer. The efficiency spikes up in the system parameter regions when the queues are not always full. The load balancer becomes useful when the arrival rate is slow enough for the severs to keep up and variability in max rounds (aka job length) overloads some servers and keeps others free. This kind of scenario can be seen in Figures 1 and 3. Its especially prominent when Lamda is 1000 [jobs / ms] with spikes of efficiency around the 2000 to 4000 max round regions. However, the efficiency significantly drops with increasing lambda for both cases of assignment.

The relative performance between the types of assignments varies with increase in Lambda. With round robin assignment, the load balancing algorithm on average matches performance without the load balancer with increasing Lambda for the 90th percentile of job response times which can be seen in Figures 7 and 8. However, with random assignment the load balance performs worse with increasing Lamda illustrated in Figures 5 and 6.

Max Rounds Impact

As alluded to the trend in previous observations with Lambda, the load balancer decreases in performance if all queues are heavily filled without the ability of keeping up. For transfer of load to be useful some queues must be able to free up while others are performing long jobs and having queued up more. Increasing max rounds fills up the queues for both assignments and degrades performance of the load balancer. This can be seen clearly in efficiency plots displayed in Figures 1 and 3. Its important to note that the load balancer is more efficient with the random assignment as the queues are not filled up in order the balancing becomes effective much earlier on. The efficiency differences are again seen through the same plots.

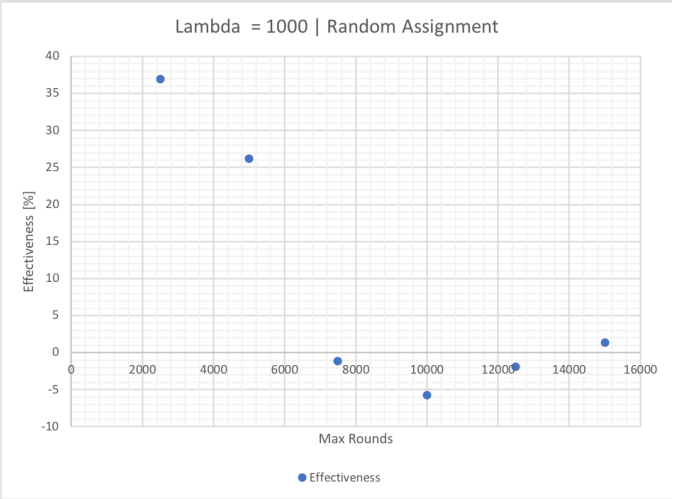
Improvements

The implementation can be drastically improved but implementing non server (queue) blocking load balancing by keeping track of a global average which changes when a job is enqueued and dequeued, as well as global sizes for each queue, which will allow for individual locking for load balancing. This will allow less time wasted for load balancing to happen as it opens up the queues for continuing to complete jobs. Futhermore, the load balacing algorithm can be improved to adapt to the history of balanced servers. This means to keep a running average of the history of how unbalanced the servers were for the past 3 iterations, and apply a multiplier for the wait time to rebalanced again. This should effectively converge to an optimal re-balancing rate based on the system parameters, almost acting as a PI controller.

Analysis wise, the load balancing algorithm needs more data in the regions of high performance to detail complete trends with data, especially synergistic ones. These are areas shown on the

efficiency graphs (Figures 1 and 3). Only initial data results are presented due to constraint in time and server overload to gather accurate data.

1 Appendix - Efficiency of Load Balancer



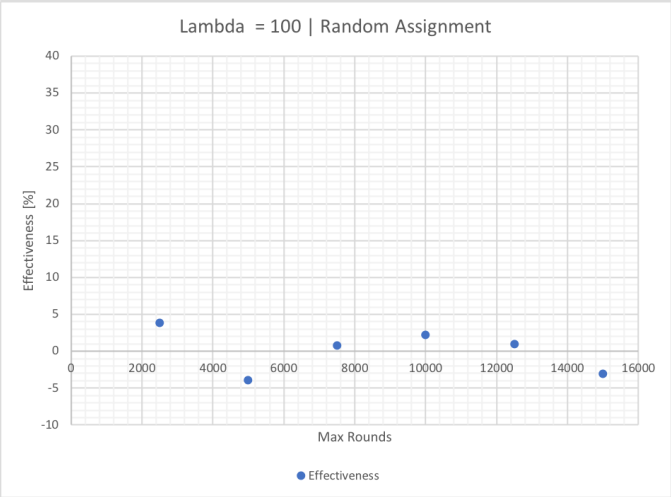
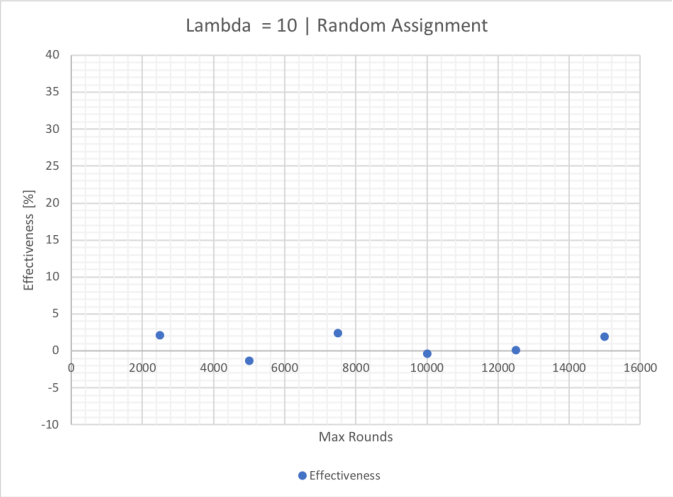


Figure 1: Logarithmic response times with and without a load balancing algorithm using a random assignment



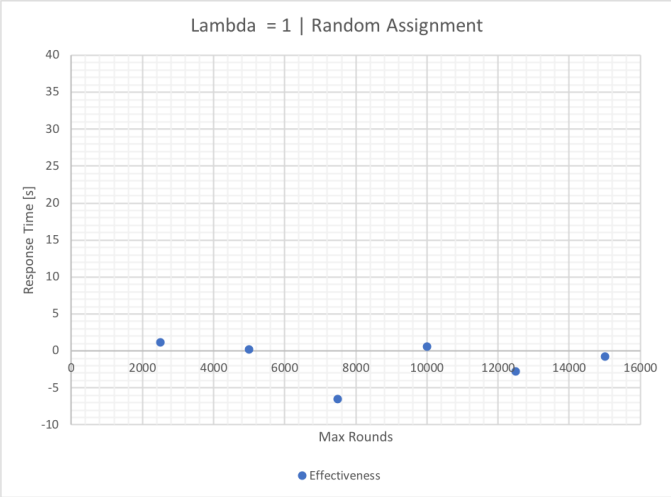
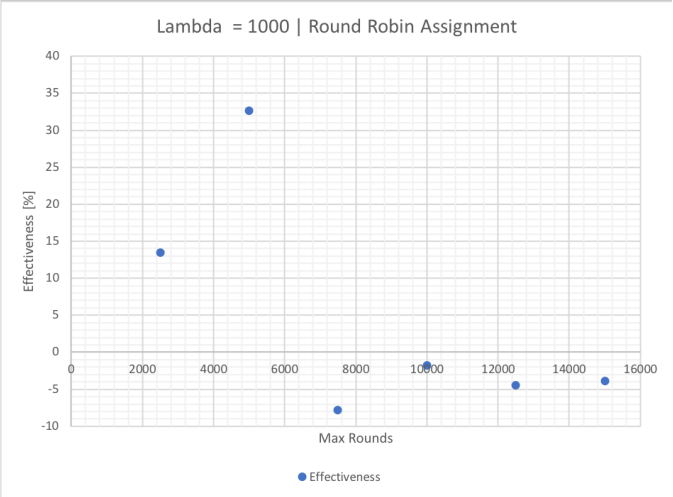


Figure 2: Logarithmic response times with and without a load balancing algorithm using a random assignment



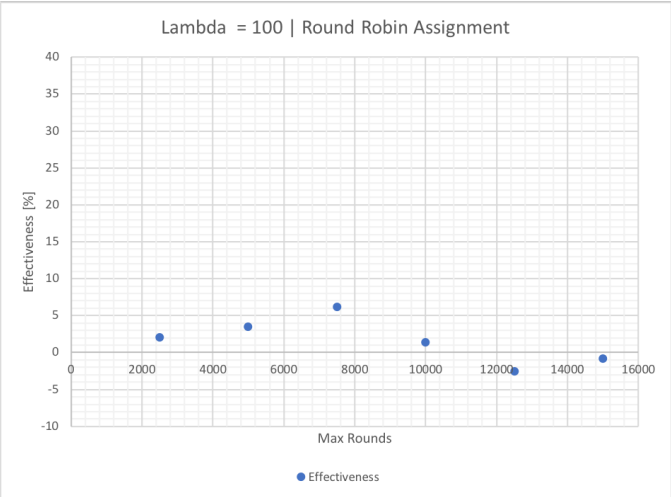
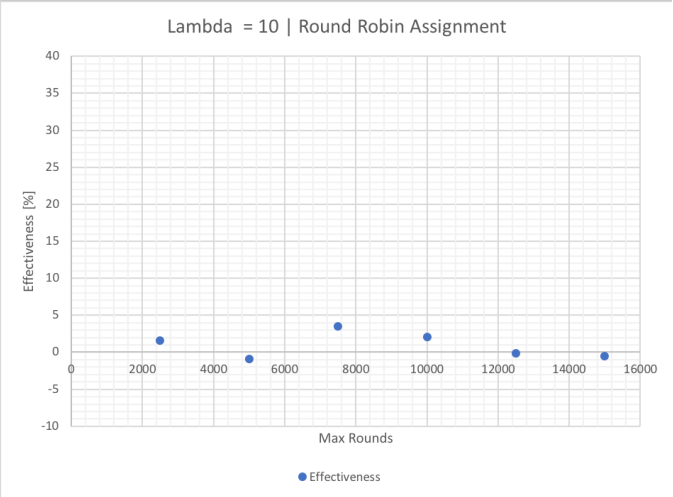


Figure 3: Logarithmic response times with and without a load balancing algorithm using a round robin assignment



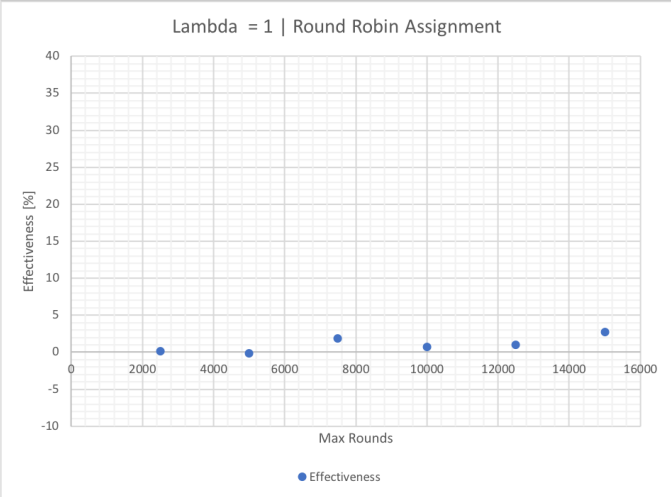
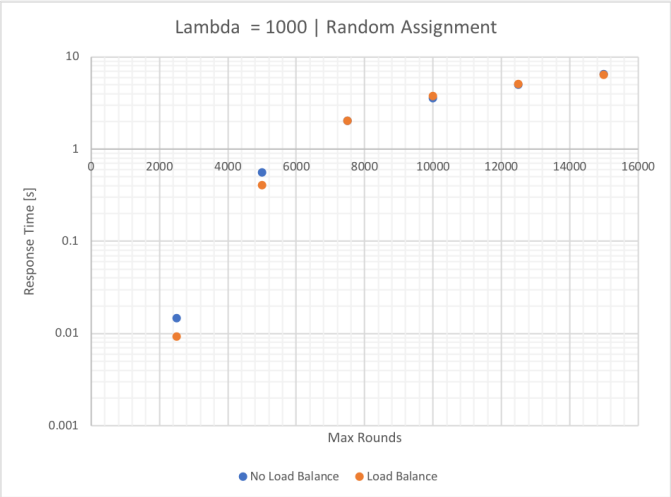


Figure 4: Logarithmic response times with and without a load balancing algorithm using a round robin assignment

2 Appendix - Logarithmic Data



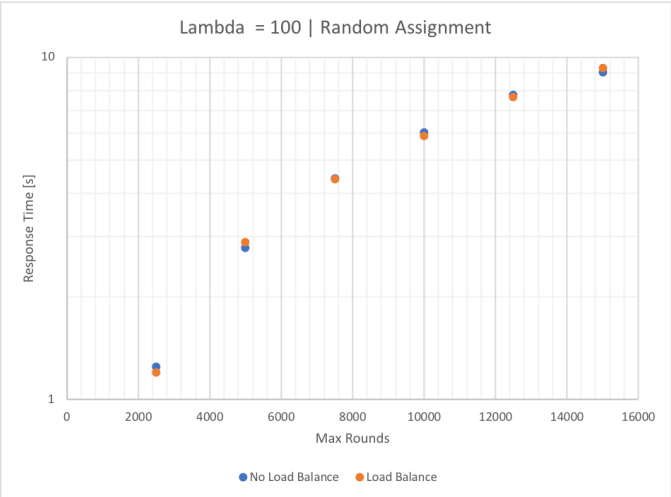
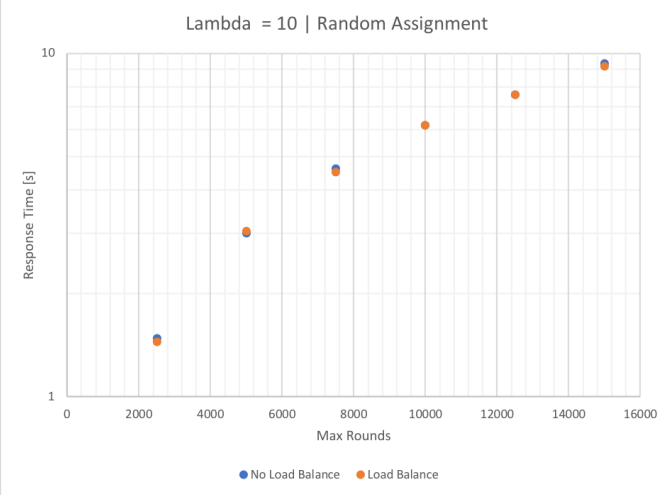


Figure 5: Logarithmic response times with and without a load balancing algorithm using a random assignment



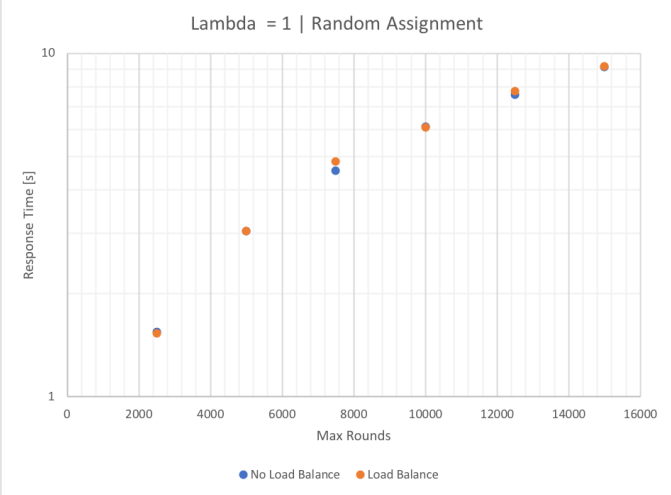
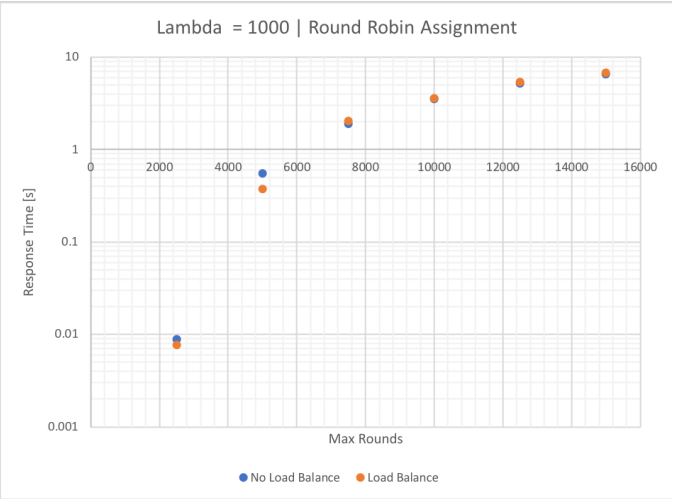


Figure 6: Logarithmic response times with and without a load balancing algorithm using a random assignment



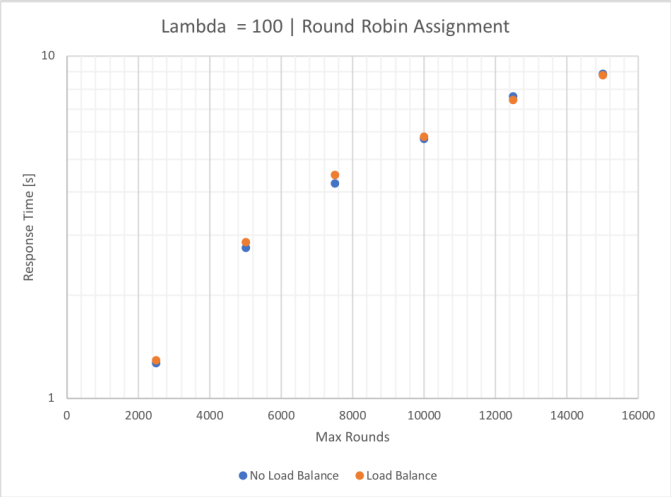
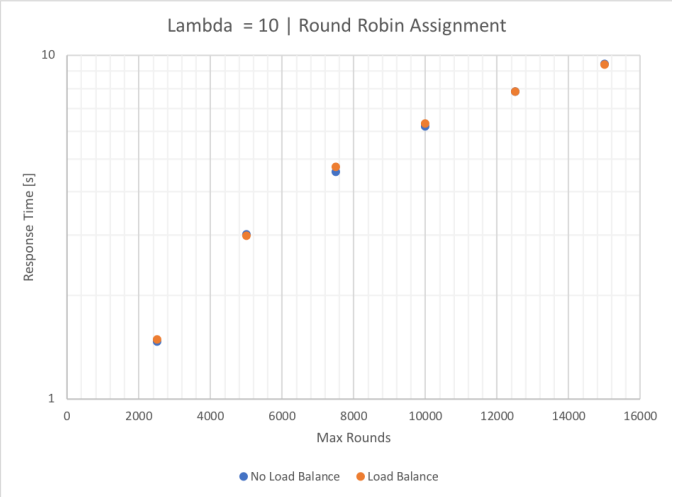


Figure 7: Logarithmic response times with and without a load balancing algorithm using a round robin assignment



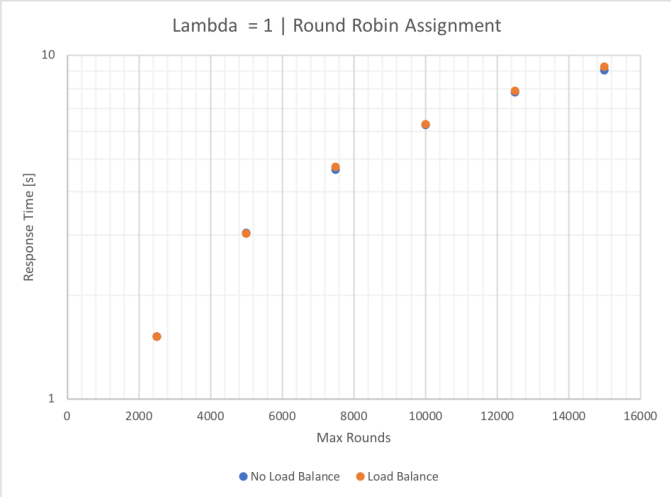


Figure 8: Logarithmic response times with and without a load balancing algorithm using a round robin assignment