# Load Balancing Algorithm for Better Response Time using Improved Queuing (BRIQ)

Dibyadarshan Hota, 16CO154 (dibyadarshan.hota@gmail.com)
Aditi Gupta, 16CO202 (guptaaditi1709@gmail.com)

May 23, 2020

## 1 Original paper

*Chien, Nguyen Khac, Nguyen Hong Son, and Ho Dac Loc.* "Load balancing algorithm based on estimating finish time of services in cloud computing." *2016 18th International Conference on Advanced Communication Technology (ICACT). IEEE, 2016.*

## 2 Problem addressed and solution

The algorithm and improvisation proposed in our paper aims to approach load balancing in a way so as to reduce the net response time and maximise throughput.

Some existing dynamic algorithms approach load-balancing by estimating the end of service time for each job, at best. This improves response time and processing time to some extent, however it does not achieve the best possible response time for all the jobs in the queue, mainly because [1] makes use of FIFO queue in the VM. Assigning every job to the least-loaded virtual machine (VM), or for that matter, the VM which can soonest finish the job, does not necessarily imply that the response time for all the jobs would be minimised.

Although there has been enough emphasis laid on how the load is distributed over the nodes, the sequence of processing the job requests in the queue still remains an unattended problem. So far, the incoming jobs in the queue were processed in a First In, First Out (FIFO) fashion.

In this algorithm, incoming job requests are stored in the queue in an increasing order of *instructionCount*, which is the number of instructions in the job. This way, we always process the *shortest job first* (SJF). However, starvation is a likely problem in this case, as a job which has a higher *instructionCount* might have to wait indefinitely because of a lot of jobs with lower *instructionCount* coming in. To counter this problem, we use *ageing* to increase the priority of a waiting job so that it gets allocated to a VM.

While there is a VM available to be allocated, a task is dequeued from the priority queue and calculations are performed for it as described in the section below to identify a VM with minimum finish time.

Please find our code here: https://github.com/aditigupta17/load-balancing-briq

# 3 Assumptions for building the solution

Our solution builds upon the fact that load balancing is performed at *two* levels - host level (known as *Virtual Machine (VM) scheduling policy*) and VM level (known as *task scheduling policy*). Furthermore, decisions can be made at either level to pick between *time-shared* (TS) scheduling and *space-shared* (SS) scheduling. Hence, the load-balancing scenarios can be comfortably demarcated into *four* scheduling scenarios as follows:

1. SS in host level and SS in VM level
2. SS in host level and TS in VM level
3. TS in host level and SS in VM level
4. TS in host level and TS in VM level

Response time for a job with arrival time *arrivalTime*, finish time *finishTime* and transmission delay *TDelay* is given as:

$$ResponseTime = finishTime - arrivalTime + TDelay \quad (1)$$

Assuming *startTime(x)* as the time when the job $x$ started, *time* as the current simulation time, *instructionCount* as the total number of instructions of a job, *core(x)* as the number of cores or processing elements required by the job, *capacity* as the average processing capacity (in MIPS) of a core for a job, we obtain following equations:

If the host scheduling policy is SS-SS or TS-SS, *finishTime* is calculated as:

$$finishTime = startTime + \frac{instructionCount}{capacity * core(x)} \quad (2)$$

If the host scheduling policy is SS-TS or TS-TS, *finishTime* is calculated as:

$$finishTime = time + \frac{instructionCount}{capacity * core(x)} \quad (3)$$

*Execution time* of a job is calculated as:

$$ExecutionTime = \frac{instructionCount}{capacity * core(x)} \quad (4)$$

*Capacity* in each case is calculated as follows:

1. VM scheduling policy is SS, task scheduling policy is SS:

$$Capacity = \sum_{i=1}^{np} \frac{Cap(i))}{np} \quad (5)$$

   where, *Cap(i)* is the processing power of the core $i$, *np* is the number of real core of the considered host.

2. VM scheduling policy is SS, task scheduling policy is TS:

$$Capacity = \frac{\sum_{i=1}^{np} Cap(i))}{max(\sum_{j=1}^{\alpha} cores(j), np)))} \quad (6)$$

   where, *cores(j)* is number of cores that job $j$ needs. $\alpha$ is total job in VM which contains the job.

3. VM scheduling policy is TS, task scheduling policy is SS:

$$Capacity = \frac{\sum_{i=1}^{np} Cap(i))}{max(\sum_{k=1}^{\beta} \sum_{j=1}^{\gamma} cores(j), np)))} \tag{7}$$

where, $\beta$ is the number of VMs in the current host, $\gamma$ is number of jobs running simultaneously in $VM_k$.

4. VM scheduling policy is TS, task scheduling policy is TS:

$$Capacity = \frac{\sum_{i=1}^{np} Cap(i))}{max(\sum_{j=1}^{\delta} cores(j), np)))} \tag{8}$$

where, $\delta$ is total job of the considered host.

The algorithm is as follows:

Step 1: Create a DatacenterBroker, maintain a status index table of the VM, current job allocated to any VM and determine its processing status. When DatacenterBroker is initialised, no VM is allocated a job.

Step 2: When there is a request to allocate a VM, DatacenterBroker will analyse status index table, estimate the time of completion of the job on each VM based on calculations discussed in the paper. The VM with the earliest completion time will be selected for the job.

Step 3: Id of the selected VM is returned to DatacenterBroker.

Step 4: DatacenterBroker posts job to VM identified by returned Id.

Step 5: DatacenterBroker notifies the algorithm about this new allocation.

Step 6: Algorithm updates the status index table of VM and job.

Step 7: When the VM finishes processing a job and notifies DatacenterBroker, it updates the job's status index in the table and decreases the job count by 1.

Step 8: While there are jobs, jump to step 2

# 4 Experimental results

Significant improvements have been achieved with respect to response time and throughput. Metrics have been logged in the table below. The experiments were run on a system having Intel Core i7-8550U CPU @ 1.80GHz x 8 processor.

Table 1: Response Time: Load Balancing using Estimated Finish Time Algorithm versus BRIQ

| **Number of Client Requests** | ***Estimated Finish Time Algorithm*** | ***BRIQ*** |
|---|---|---|
| 10 | 14.944537 | 14.944537 |
| 100 | 38.853931 | 36.879389 |
| 1000 | 434.767279 | 313.244709 |
| 10000 | 4549.779616 | 3205.598167 |
| 100000 | 45868.511618 | 32449.084248 |

*Response Time reported in milliseconds

Table 2: Throughput: Load Balancing using Estimated Finish Time Algorithm versus BRIQ

| **Number of Client Requests** | ***Estimated Finish Time Algorithm*** | ***BRIQ*** |
|---|---|---|
| 10 | 17.811189 | 17.811189 |
| 100 | 36.130435 | 31.257143 |
| 1000 | 438.5 | 252.113043 |
| 10000 | 4551.148936 | 2551.953271 |
| 100000 | 45917.8 | 26019.708333 |

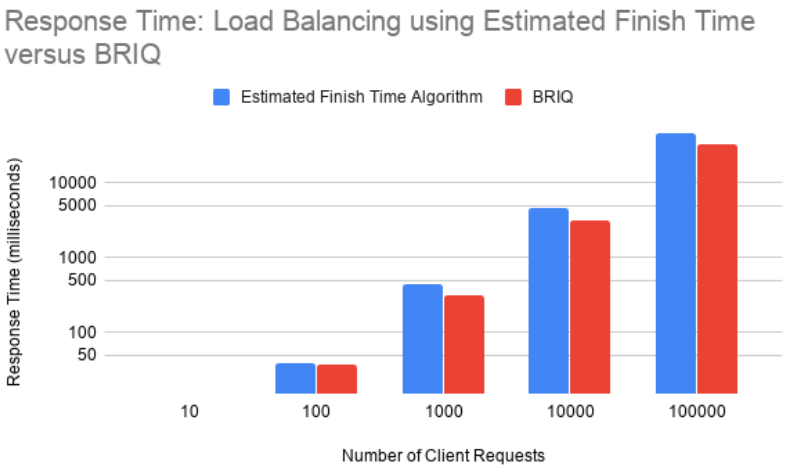


Figure 1: Response Time: Load Balancing using Estimated Finish Time Algorithm versus BRIQ

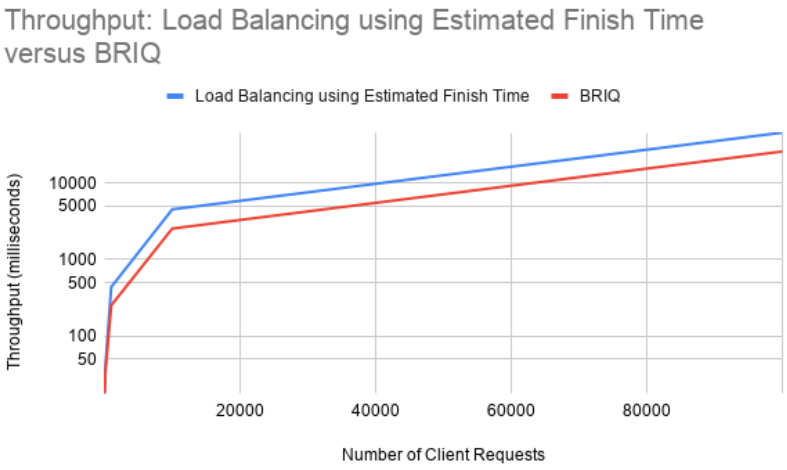


Figure 2: Throughput: Load Balancing using Estimated Finish Time Algorithm versus BRIQ

# 5 Related problems which are not addressed

1. This algorithm does not address the issue of degree of imbalance which measures the imbalance of loads among the VMs.

2. Another issue which is not addressed in this algorithm is regarding preemptive jobs. Right now, every job gets completed in one burst. There could be a room for better throughput if preemptiveness is allowed.

# 6 Limitations of the solution implemented

The overhead of using a priority queue should be minimized. Priority queue can be implemented using many different heaps like Fibonacci heap, binomial heap, binary heap, 2-3 heap, etc.

All of these data-structures have different complexity trade-offs. For the purpose of a load balancer, these data-structures must be analyzed and improved upon for optimal performace.

# 7 Future work proposed

1. Tuning the algorithm parameters and improving the heuristics to get better response time.

2. Integrating elements of other various load balancing algorithms (for example ACO optimization, genetic algorithms, etc) into the proposed load balancer in order to improve results.

3. This algorithm achieves better results in case of SS-SS. Further improvements can be done with regard to remaining three sceanios.

4. There does not exist a load balancing scheme which is suitable for all applications and all distributed computing systems. The choice of load balancing mechanisms depends entirely on applications and hardware specifications. Such a dependency needs to be relaxed.

# 8 References


1. N. K. Chien, N. H. Son and H. Dac Loc, "Load balancing algorithm based on estimating finish time of services in cloud computing," *2016 18th International Conference on Advanced Communication Technology (ICACT)*, Pyeongchang, 2016, pp. 228-233.

2. Jasmin, J., Bhupendra, V., âEfficient VM load balancing algorithm for a cloud computing environmentâ, *International Journal on Computer Science and Engineering (IJCSE)*, 2012.

3. J. Ni, Y. Huang, Z. Luan, J. Zhang and D. Qian, "Virtual machine mapping policy based on load balancing in private cloud environment," *2011 International Conference on Cloud and Service Computing*, Hong Kong, 2011, pp. 292-295.

4. K. Nishant et al., "Load Balancing of Nodes in Cloud Using Ant Colony Optimization," *2012 UKSim 14th International Conference on Computer Modelling and Simulation*, Cambridge, 2012, pp. 3-8.

5. T. Deepa and D. Cheelu, "A comparative study of static and dynamic load balancing algorithms in cloud computing," *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, Chennai, 2017, pp. 3375-3378.

6. Lu Kang and Xing Ting, "Application of adaptive load balancing algorithm based on minimum traffic in cloud computing architecture," *2015 International Conference on Logistics, Informatics and Service Sciences (LISS)*, Barcelona, 2015, pp. 1-5.

7. K. D. Patel and T. M. Bhalodia, "An Efficient Dynamic Load Balancing Algorithm for Virtual Machine in Cloud Computing," *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, Madurai, India, 2019, pp. 145-150.

8. L. Zhu, J. Cui and G. Xiong, "Improved dynamic load balancing algorithm based on Least-Connection Scheduling," *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*, Chongqing, China, 2018, pp. 1858-1862.