Kala Venkata Sri Sai Bhargav S20190010077

OPTIMIZING PERFORMANCE WITH LOAD BALANCING ALGORITHM TO REDUCE LATENCY, THROUGHPUT AND PROCESSING TIME ON CLOUD COMPUTING

Principal, G. M. B. OPTIMIZING PERFORMANCE WITH LOAD BALANCING ALGORITHM TO REDUCE LATENCY, THROUGHPUT AND PROCESSING TIME ON CLOUD COMPUTING.
Publisher : Aut Aut Research Journal
ISSN NO: 0005-0601

Motivation of the paper:

Cloud computing is a new generation emerging technology which comes with many challenges. Though the technology is in great demand and meets the needs of the people efficiently, the user experience is constantly threatened by problems such as response time for the user and the bottlenecks created by the rapid growth of data and usage for the cloud providers. Thus, for the efficient work of the VMs and to improve performance for the end user, this paper proposes a load balancing algorithm which decreases the response and processing time thus improving the efficiency.

Objective of the paper:

The objective of the paper is to come up with a new algorithm with a throttled set of change of rules which performs better than old-school scheduling algorithms and thus decreasing the response time and increasing the efficiency.

Major contribution:

The contributions made in this paper are:

- A new algorithm is proposed in context of load balancing which is an improvement to the old school Round robin scheduling algorithm and the throttled algorithm

- The new algorithm is deployed and tested against various existing algorithms and real world results are verified.
- An approach as to how to deploy this algorithm is also discussed.

Description of the original algorithm:

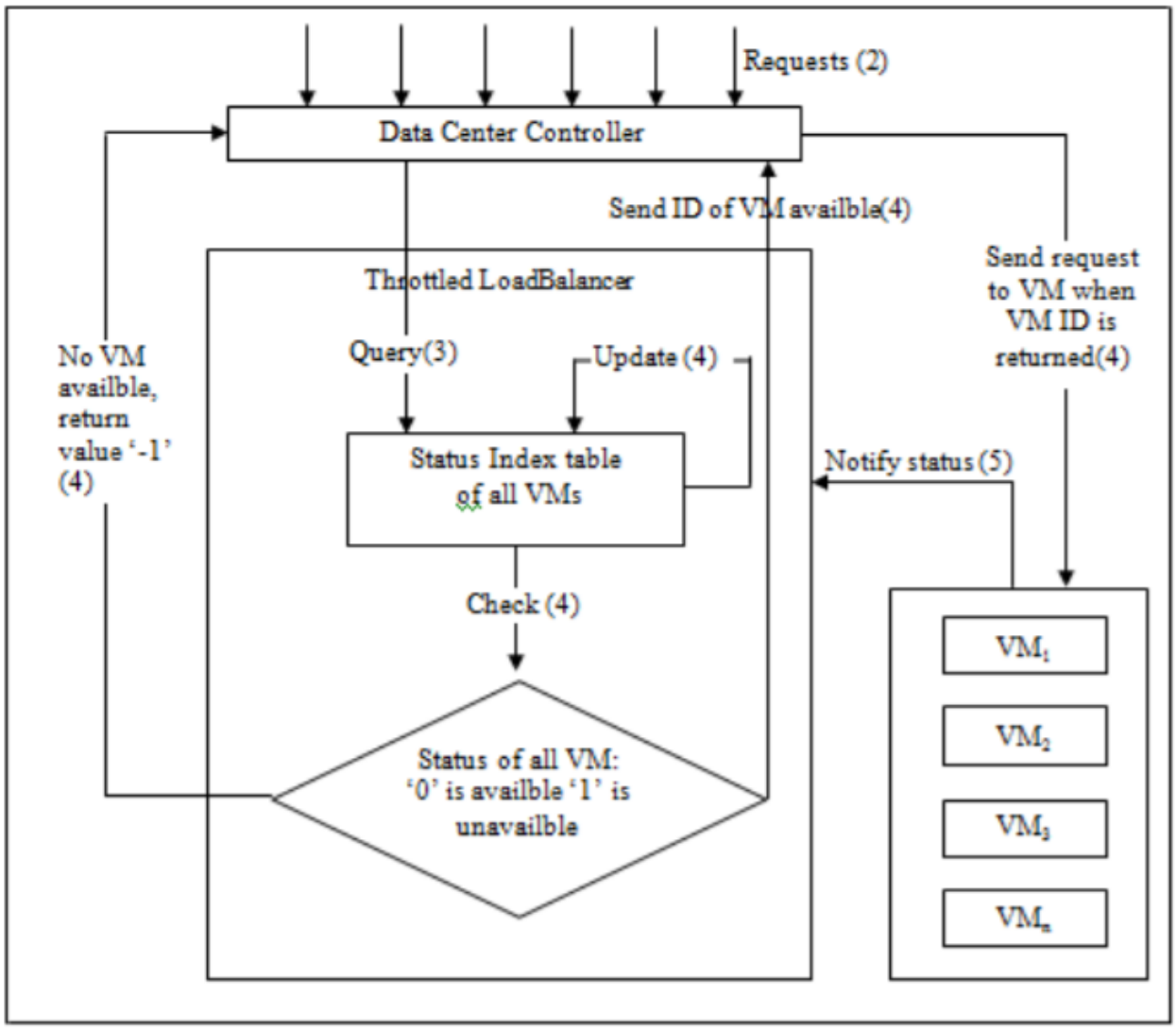
1. **Round Robin approach**: (Originally used algorithm)

   This is the original approach used back then which is basically dependent on the quantum idea. This basically tries to distribute the burden to the VMs in circular rotation fashion. This basically runs on the idea that all the VMs are of the same computational power and thus the task is equally subdivided.

   CON: Data centres with VMs of bigger computational capacity get wasted and thus there is a wastage of resources as the algorithm considers all VMs to the of the same capacity and thus distribute equally.

2. **Throttled Set of rules**: (Originally used algorithm)

This architecture is as follows:



1. Throttled Load Balancer(TLB) keeps an index where 0 represents free VM and 1 represents occupied VM. All VMs are at 0 at the start.
2. Information Middle controller(IMC) gets a new request.
3. Statistics Middle Controller(SMC) asks the TLB for a new undertaking.
4. TLB now starts checking which VM is free from the top of the desk and returns the first free VM.

    If a free VM is found:

        1. The TLB sends the ID of the VM to SMC.
        2. A request is sent to the VM and the TLB is notified of new allocation.
        3. The TLB now sequentially goes and updates the index and waits.

    If a free Vm is not found:

1. The TLB returns -1
2. The request is arranged, that no free VM is available.

CON: The TLB has to sequentially go and check the status of every VM to allocate and then sequentially go and update the status if a VM is occupied and when it has completed the work.

Imagining a huge number of VMs, it is time consuming task to pass through the entire list to update the index of the status of the VM.

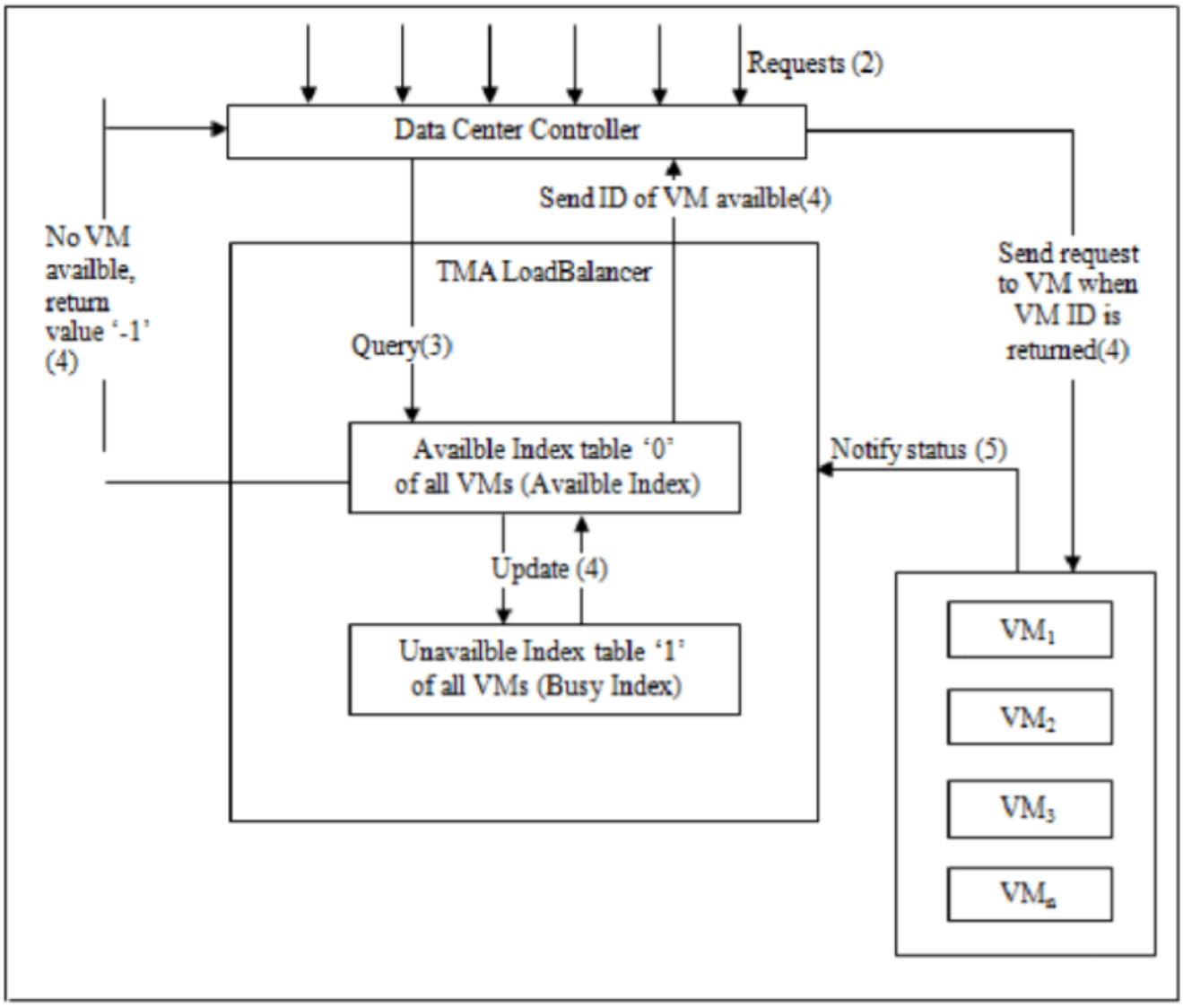
Description of the proposed approach:

Throttled Modified Algorithm: (Proposed Algorithm)

The architecture is as follows:

Here, instead of maintaining a single index to store the status of the VMs, we store two indexes. One representing Available VMs and the other representing busy VMs. The Available VMs are denoted with 0 and the occupied VMs are denoted with 1. All VMs are originally placed in the available index.

The sequence of steps during allocation are:



1. Data Centre Controller(DCC) gets a new request.
2. DCC requests the TLA for allocation of machines.
3. The TLA detects and sends VM-ID (VM) from the top down in the "Available Index" table of the Data Center Controller.
4. DCC sends the request to that VM and updates the TLA about the allocation of the VM.
5. The TLA updates the status of this VM as busy and waits for the next request.
6. Once the VM completes work, the DCC is informed which inturn updates the TLA to update the Available Index.
7. If multiple requests are to be processed, the VMs are allocated until the “Available Index” is empty and then -1 is returned.

BENEFIT:

With our proposed algorithm (TMA), it will be possible to detect the VM available (status '0') with the size of the table "Available Index" more flexible than the Throttled Algorithm. This improves the performance of the system by vastly reducing the number of passes to be made.

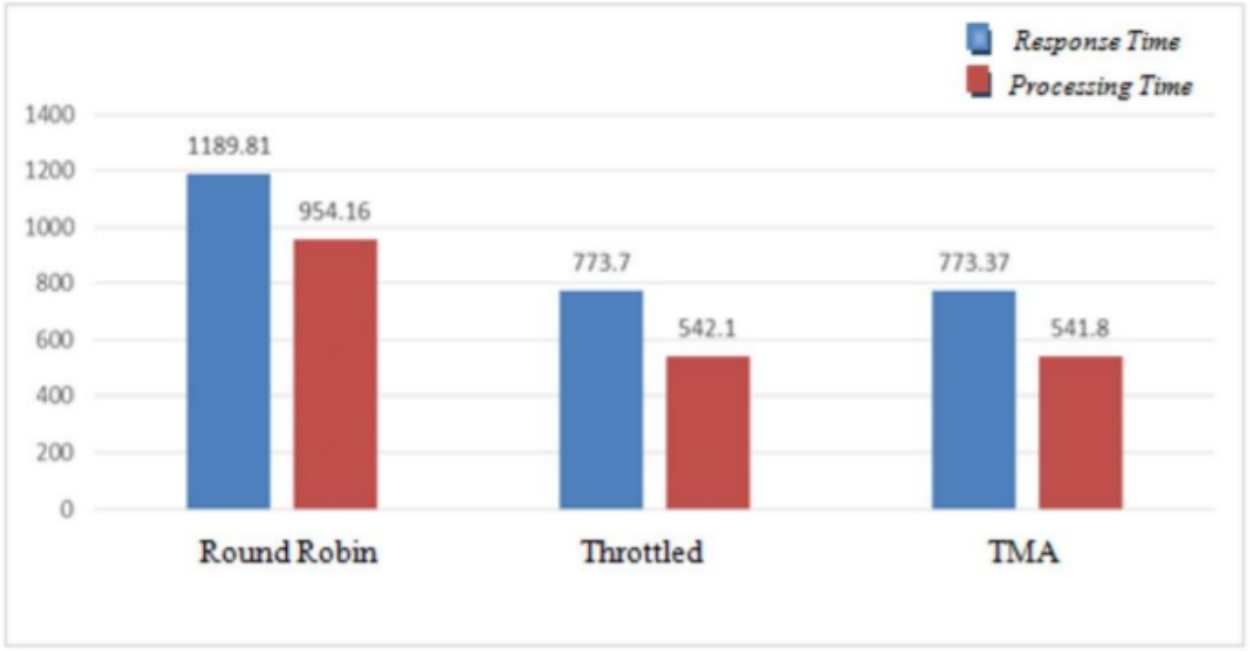
Discussion of the experimental testbed and results:

These algorithms were now conducted in similar User and Vm configuration settings, similar Data Centre configuration settings and similar internet configuration settings. It shows how different algorithms work and how much time they are taking for producing a response and to process the request.

The experimental testbed used is:
The authors have simulated 6 User bases that correspond to six zones with a specific time zone, and most users use the app in the evening for about 2 hours after work. Every 5 minutes, each user sends a new request while online. These parameters are configured inside the main configuration tab of the configure simulation class. All VMs are configured the same way.

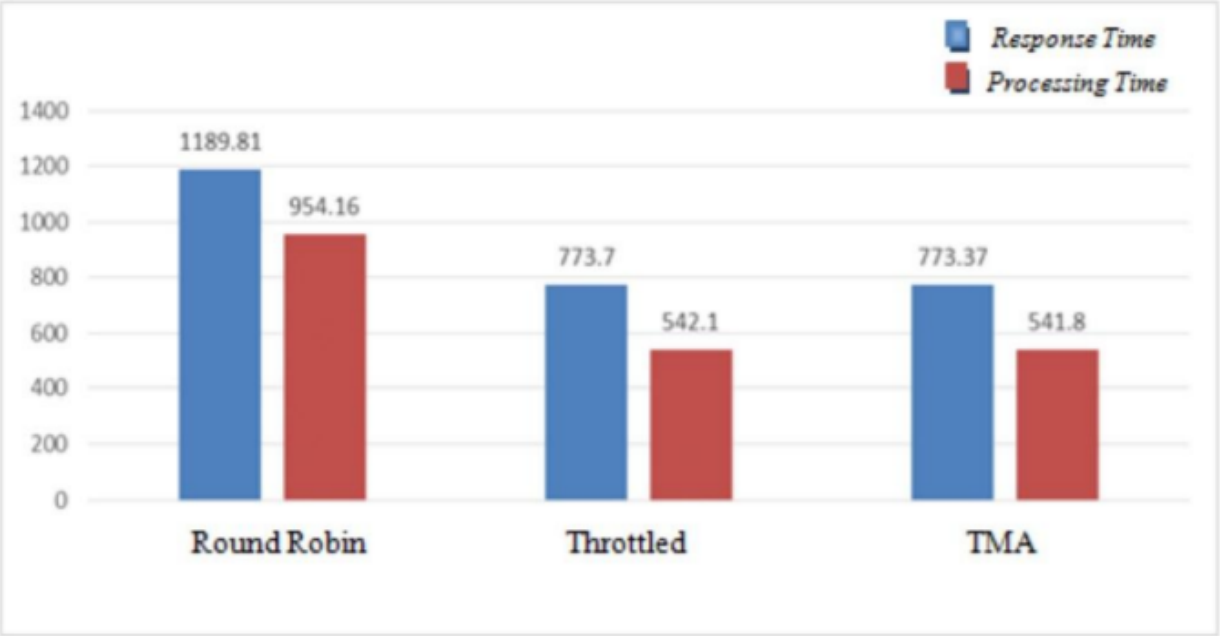
Scenario 1:

Simulated with 20 VMs:



Scenario 2:

Simulated with 50 VMs:



The result from these is that It helps us to see that with the TMA algorithm, the **variety of requests that need to queue has decreased**, in addition to the processing time of the data middle and the **response time of the machine is stepped forward than the two algorithms**. Because of this the tma set of rules has better load balancing than the throttled and spherical robin algorithms.

Strengths:

- The time required to find and allocate a free VM now reduces as we don't have to check the entire list of busy VMs.
- Considering a very huge number of VMs (in terms of millions), the time required to allocate resources reduces rapidly thus yielding in **less response time.**
- The update sent regarding the allocation or de-allocation of the VM need not be queued along with the other requests,and as all the requests are distributed between the tables, queuing does not occur, thus resulting in **better processing time.**

Limitations:

- Since we are removing the index of a VM from Available index and pushing it to the Busy index and vice versa, the failure of either of the indexes after the removal from the first index happens, would result in the entire loss of the index of the VM which leads to wastage of resources.
- Instead of updating a single table, we need to update two tables everytime a VM is allocated or returned. When a VM is allocated, we need to remove it from the Available index and push it in the busy index, which adds overhead and is always prone to loose.

- Also we now need to maintain the metadata of two different tables compared to a single table along with the extra pointers and registers the TLB will need, to parse different tables for different needs thus needing extra space.
- Instead of checking the single table, we now need to differentiate the requests into different groups based on if they are directed to the Available index or the Busy Index which needs extra computational power and adds an overhead thus making the architecture much complex.
- As the inter dependencies increase, the chances of failure increase, failure of any one index or the failure of expected behaviour of any one operation can mess the whole system up.