Performance evaluation of proposed load balancing algorithm with unstable concurrent programs

Chanintorn Jittawiriyanukoon
Assumption University, Thailand

Article Info

Article history:

Received Dec 05, 2018
Revised Jan 21, 2019
Accepted Feb 27, 2019

Keywords:

Internet of thing (IoT)
Load balancing
Queueing network
Scheduling mechanisms
Simulation
Task partitioning
Unstable concurrent programs

ABSTRACT

IoT is the succeeding cohort of the digital computing environment. A swift progression in the IoT deployment and its applications are on the rise. Improving load balancing mechanisms induces healthier performance of the internet based computing as higher number of users can be comfortable. Implementing full services for tasks with unstable concurrency is an uphill process. One of the encounters allied with this administration is the task partition among the applications, regularly referred as concurrent programs. Through load balancing not only resources are equally utilized but also concurrent job's response time can be promoted. Therefore, in this paper the widely used load balancing algorithms are investigated and yet the proposed algorithm is introduced. Simulation is employed in order to compare the performance metrics such as mean queue length, utilization and throughput between the recommended and existing algorithms. The proposed algorithm confirms the load balancing and outperforms when processing unstable concurrent programs.

Corresponding Author:

Chanintorn Jittawiriyanukoon,
Assumption University,
Samut Prakan, 10540, Thailand.
Email: pct2526@yahoo.com

1. INTRODUCTION

There is no designation for digital computing, but it is crew of distributed processors which affords the internet based services on demand [1]. The applications are collection of software users deliver for their requests. Typically the computing has the major component such as data service center; this is assembly of processors hosting diverse applications. This may vary at a large diversity from the user's point of view. Presently, an impression called virtualization [2] is employed to run software which tolerates concurrent programs over virtual machines. The distributed virtual platforms are prompt throughout the internet hosting applications. But while involving concurrent applications, the user will transparently sense that he is using it from his own computer.

Börger, *et al* [3] explain an independent language for the transactional operator and controller which are concurrent programs with public locations and return their observance to the dissolution condition. They verify the accuracy of concurrent programs running through the transactional processing under serialization. The abstract state of the virtual machine is specified for both operator and controller. It defines controller application to an extensive choice of programs and especially offers the option to make use of a plug-in as clarifying by abstract state in the concurrent system. Another research has reached to indulge a memory for concurrent programs in terms of the abstraction approach [4]. Khan, *et al* [5] define a load balancing scheduling technique used in parallel processing system. The idea is to keep all distributed processors equally busy all the time in order to avoid a wasteful idle time. They also demonstrate a load balancing algorithm for distributed system by means of the load imbalance factor. The algorithm per se is based on the optimization approaches such as min-max and max-max methods. Raghu and Manjunatha [6] present a development of smart grids to control structures of power system using a network computing model. They achieve a load balancing between the system performance and the load shedding. The load balancing is applied and helpful for the conclusive decision on the selection process.

Concurrent computing is a processing model where parallel processors perform requests harmoniously for superior performance. Concurrency explains something that occurs simultaneously as other else. Original programs are divided down into subtasks which are routed to distinct processors to execute at the same time together, instead of serial fashion as they are fulfilled by a single processor. Concurrent programming is said to be compatible with parallel processing. In mobile applications, concurrent programs are batch processes which run and retrieve data from the database system. The concurrent program comprises of a set of requests, allowing the fair processing of each single request independently but executing on fair-parallel multiple processors. Automatic verification of the concurrent program coded in low-level language can be found in [7]. There are several researches on load balancing for the cloud computing [8]. Load balancing with partitioning model in cloud environment is presented in a paper written by [9] who introduce the algorithm based on the game theory to improve the load balancing efficiency. However, the load balancing concept in partitioning clouds is an issue which needs innovative strategies to cope with various changes. Besides, there are load balancing algorithms, such as Leaky bucket, Round Robin, Throttle and Ant Colony [10] in order to optimize the efficiency of distributed clouds.

Bouajjani *et al* [11] announce a method for comparing two dependent concurrent programs, in terms of data flow and cross-thread interferences, under an abstraction that identifies any differences in the limit. The dependence between these abstractions regarding a change of the program which indicates an abstraction is following a regression-free concurrency. Sung *et al* [12] progress an approximate method for calculating synchronization of the two concurrent programs. This approach is brisk as, instead of counting on weighted model, it empowers a datalog-based analysis to calculate differences of the data flow. A web-based semantic is presented in [13] for the knowledge in medical area. A context-based model for multi-threaded programming can be found in [14]. Especially, a proof-rule whose ground involves authenticating serialized function of concurrent programs is developed in the paper [15]. It avoids inducing multi-threaded structure as clearing premises, in an automatic fashion for non-concurrent programs.

The intent of the paper is to assess the efficiency of load balancer with general concurrent programs using a simulation [16]. In the investigation, a general characteristic of concurrent programs is discussed in the Section 3. In order to involve with these concurrent programs, the proposed algorithm which can resourcefully handle the load balancing is introduced then results from simulation are rounded to validate the accuracy and effectiveness by comparing to those results from existing algorithms. Finally, other performance matrices of the proposed method are valued.

The state-of-the-art operating system is designed to handle substantial multi-processing units concurrently. This develops multitasking capabilities for operating systems like Linux Mint or Microsoft Windows. In multi-tasking situation [17], the computing is the concurrent maneuver of manifold processes over a while. Fresh tasks can start over prior to the end of others, rather than waiting for the overall completion. As a result, multi-tasking involves parallel processing of various tasks instantaneously, rather than it consents a single task to advance in sequential fashion. Multi-tasking is a public characteristic of contemporary operating systems. It remunerates a resourceful deployment of computer resources; while an application has to wait for input/output cycle to end, the CPU then can still avail for alternative application. In a time-sharing alignment, heterogeneous processes consume a single-and-the-only processor as if it was committed to the use, while the only processor is servicing users by multi-tasking them through individual application.

However, different size of concurrency roots a performance pressure on load imbalancing [18] as it is challenging to manage a load balancing. Not to mention concurrent tasks will lead tedious problems such as locality and linearizability as specified in [19]. Moreover, concurrency also develops storage and scheduling problems in the distributed system [20]. As such, the experiment in this paper copes with distributed problems of mutable concurrent tasks, particularly addresses how to police unstable concurrent tasks then balance the load in terms of storage administration. The proposed algorithm computes the adaptive load amount to suit the bandwidth of sliding windows. After that, the proposed algorithm applies the policing mechanism in order to balance the load in terms of storage management. The existing three algorithms, namely, Leaky Bucket (LB), Round Robin (RR) and Throttle (TH) are applied for inspecting the effectiveness of the proposed algorithm. The experimental results using simulation are piled to validate the accuracy between existing three and the proposed algorithms. The layout of the paper is as follows. In the Section 3, a characteristic of unstable concurrent programs is outlined. Section 4 explains the load balancing algorithms as well as the proposed algorithm. Section 5 presents experimental results and the following section is the conclusion of the paper.

2. UNSTABLE CONCURRENT PROGRAM CHARACTERISTICS

Scheduler [21] is critical in load balancing on performance of parallel systems. It becomes tremendously provoking on heterogeneous processing powers, particularly when awkwardly concurrent programs are measured. For example, in case of variable concurrent applications, that is, a program made of an assembly of autonomous but equal tasks, to be scheduled on a parallel platform. Even though modest, this category of application is distinctive of complicated problems, including load imbalancing, multi-threads and computation overhead. If the parallel servers are uniform, i.e., identical channel bandwidths to/from indistinguishable CPUs, then a simple greedy algorithm [22] can achieve an optimum throughput. On the contradiction, if diverse CPUs, connected through changeable channel speeds, then the mentioned algorithm is unlikely to fulfill as it is vital to route to which appropriate servers.

This division describes the characteristic model of unstable current programs utilized in the experiment. First, programs so entitled intrinsic tasks will exhibit number of autonomous concurrency. The processing power comprises of parallel servers and an unidentified server (US). The intrinsic task attends the US at no frill term, but intrinsic tasks are fragmented into auxiliary tasks according to the number of variable concurrency. The auxiliary tasks from the associated intrinsic task are called kin. All kin emerges parallel servers. It is completely autonomous amongst kin, but the consequence of queue while waiting for service. The service time after the presence observes the exponential fashion. If a kin ends the execution, it revisits to the US in order to pause for the conclusion of all kin as depicted in Figure 1. If all kin is done, they are fused into the intrinsic task. This marks the synchronization of the kin and proceeds to subsequent step. The interval from attending the intrinsic server to revisiting is named expectancy. If an intrinsic task (C_m) is fragmented in the US into C_{mn} mutable auxiliary tasks then it cyclically exhibits the concurrency, C_{mn}. Note that C_{mn} revises repeatedly due to variant after the expectancy. The variable concurrency matrix of any intrinsic tasks (C_m) contains a row of mutable concurrency vector representation is listed in (1). The C_m matrix is anticipated to be a deterministic set. A component C_{ij} is set of the mutable concurrency whenever $\{C_{ij} \geq 0,\ 1 \leq i \leq m;\ 1 \leq j \leq n\}$. An example of variable concurrency vector (VCV) for two intrinsic tasks (C_1 and C_2), where C_1 = [1 2 3] and C_2 = [4 2 0] is shown in Figure 1.

$$\begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_m \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{1}$$

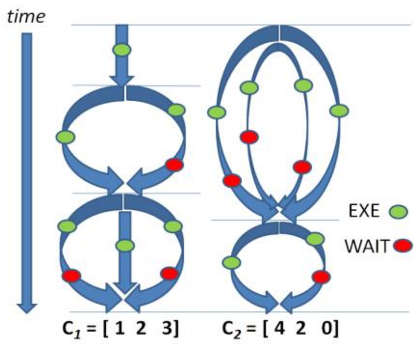

Figure 1. Example of variable concurrent programs

3. PROPOSED LOAD BALANCING ALGORITHMS

With the advent of cloud environment, the impression of balancing the load between the clouds, cloud servers and datacenters has multiplied vast attractiveness. That is the reason why lots studies are currently focusing on to smooth the load balancer, particularly at peak load hours. Load balancing is a crucial prominence in cloud computing, especially the hybrid clouds in order to cope with the service level agreement (SLA) and the quality of service (QoS). Load balancing algorithms are mostly characterized into static and dynamic ones. Static load balancing algorithm is the concept of fixing the balancing criteria from the beginning such as leaky bucket or round robin algorithms. Dynamic load balancing algorithm is to choose a proper node for any requests by firstly collecting nodes information such as CPU utilization factor or current queue size. Thus, it does not only observe the situation but also choose the right node to dispatch accordingly.

3.1. Leaky Bucket (LB)

The load balancer stabilizes the total requests in the way that it polices and shapes the traffic. The LB algorithm [23] is a manner of provisionally keeping requests and shaping them into a set-rate dispatch of packets in a high-speed network. The LB is also utilized to govern the metered-bandwidth of any connections to avoid going over the limit for a period of given time, hence preventing an extra charge. The algorithm per se functions analogously to such a manner of a leaky bucket with liquid. The LB holds the packet up to its size. Packets are only discharged from the bucket at a set-rate and packet amount. In case of overcrowding, the packet is deliberated to be non-conformant.

3.2. Round Robin (RR)

The load balancer in RR [24] schedules a request by deliberating the fairness, since it takes turn to allot each task in the queue. Each task is proceeded to take a service within the allotted time unless it is preempted then re-allocated to the queue's tail in order that the subsequent task from the queue can take on the server instead. RR scheduler is a process used by applications that serve multiple users who request for resources. It leverages requests in a rounded first-come-first-served (FCFS) fashion and shuns the priority so that applications fairly use the resources. It is the classical, modest, fairest and broadly used algorithms, as it is artless to apply and there is no concern about priorities, only a FCFS scheme and an allotted service time for each resource. This also solves the problem of bandwidth hogging where a task cannot utilize resources for a lengthy time as the preemption mechanism is applied.

3.3. Throttle (TH)

In throttled process [25] the user first requests to the datacenter for a fit server to perform the request. The load balancer, at any set-time, dispatches certain tasks to the server. All requests are queued up first if receiving high number of request (busy server), and whenever any servers become available then task gets a processing. The algorithm applies traffic policing to limit the flow rate and fix number of traffic load. The balancer quantifies the traffic-flow conformance. If the queue is overfilled beyond the capacity then subsequent arriving tasks are non-conformant.

3.4. Proposed Algorithm

The proposed algorithm is designated in the following division. The intrinsic program is fragmented into the concurrent task (C_{mn}) at the US. The load calculation to insure C_{mn} in every dispatch levies the lucrative computation of every subsolution. To reach triumph at each fractional step, the proposed algorithm only envisions the subsolution result from that step. The decision of each step the proposed algorithm concludes complies with destined buffers and the C_{mn}. This reflects a global consequence to attain the fit solution and is ample to settle an ultimate goal. It is like a way of anticipating one step ahead. The proposed algorithm requires straight decision rule as it ponders all subsolutions at each partial step. The computation cost is $O(mn)$. The proposed algorithm is digested as depicted in Figure 2.

Presently to quantity the accomplishment of applications in an economical market, only the GUI is not adequate. Poor information management affects the brand, revenue and CRM. This phenomenon shifts the business centric to service level agreement (SLA) in order to employ technology for benefits of business improvement. The objective of the paper is to balance the information load over the scalable clouds as shown in Figure 3, without conceding on scalability, cost and performance metrics. The load balancer helps tune up the load balancing in clouds. The proposed algorithm polls space buffer at the destination to agree concurrent requests from users. Steps are firstly checking concurrency from clients. For each concurrent request, check buffer requirement threshold. If it does not meet the requirement then eye on suitable but available spaces. Before migrating the request, to ensure the load on that datacenter must be less.

4. RESULTS AND DISCUSSION

The simulation model based on queuing network is employed and processing units referred as server 0-5 are labeled. Server 0 is a US with zero serviced time. All queue disciplines are set to be first-come first-serve (FCFS) basis except for the US server. The server 1 represents a balancer node (BA), while the remaining servers, server 2 to 5, denote data center representative servers with the similar branching probability. These four data centers (server 2 to 5) are joined in parallel fashion. These progam-dependent service time distributions for server 2 and 3 (data center, DC1 and DC2) follow exponential distribution with mean of 0.1, 0.2, 0.3, and 0.4 second for C_1, C_2, C_3, and C_4 respectively. The progam-dependent service time distributions for server 4 and 5 (data center, DC3 and DC4)) observe exponential distribution with mean of 1, 2, 3, and 4 second for C_1, C_2, C_3, and C_4 correspondingly. The queuing model for the experiment is shown in Figure 4. The arrival rate of all user programs at the US server shadows Poisson distribution with a mean of 1 intrinsic task per second.

Proposed Algorithm

```
Require: Mutable concurrency matrix $[C]_{mn}$ with $m$ rows and $n$ columns
Ensure: $[C]_{mn}$, S = all potential solutions in each computation step = $\{S_1^t, \ldots, S_y^t\}$,
        $B_y^t$ = available space of the buffer $y$, $O_y^t$ = candidates in each computation
        step, $P_r$ = a premium solution where $P_r(S_y) \geq 0$ and $S_y \in S^t$

for $J = 1$ to $n$ do
        $O_y^t \leftarrow 0$
 for $k = 1$ to $y$ /** All candidate seats **/
        $F = \arg\max_{S_y \in S^t} P_r(S_k)$
        /** Solution $F$ and corresponding $B_y$ **/
        end for

        for $r = 1$ to $m$ /** Choose a task to fit a buffer **/
        $\Delta_r = F - C_{rj}$
        If $\Delta_r \geq 0$ then
        $O_y^t = \arg\min_{S_y \in S^t}(\Delta_1, \Delta_2, \Delta_3, \ldots, \Delta_m)$
        ** A new best for this computation step **/
        end if
        end for
        Return $O_y^t$
end for
```

Figure 2. Proposed algorithm

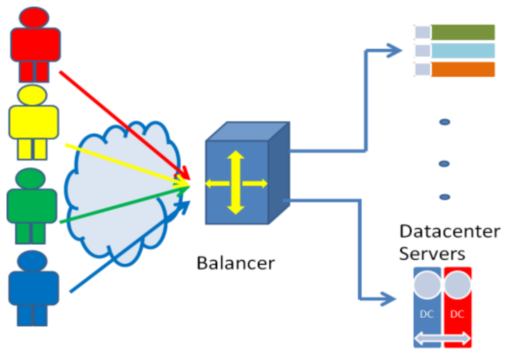

Figure 3. Proposed architecture

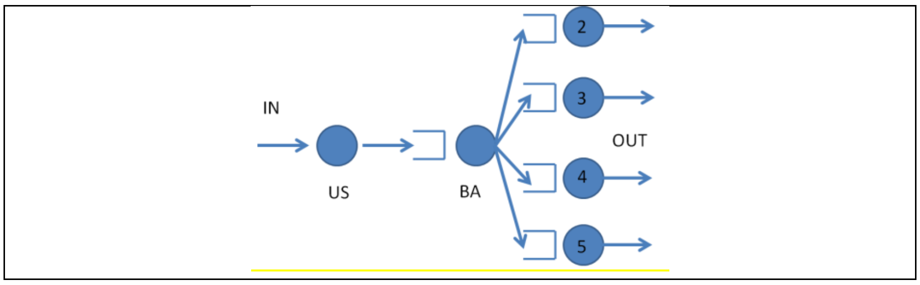

Figure 4. Simulation model

The variable concurrency vector (VCV) from four users for the experiment of the load balancing is depicted in (2). The simulation [16] is utilized to collect specific parameters (such as mean queue length (MQL), throughput (THR), mean waiting time in queue (MWT), utilization (UTL), etc.). Load Balancing Simulation Results from LB, RR, TH, PR as shown in Figure 1 to 4.

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \begin{bmatrix} 2\ 4\ 6 \\ 3\ 8\ 0 \\ 1\ 3\ 7 \\ 5\ 9\ 11 \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{2}$$

The resulting is the sequence of highest to lowest performance of the algorithms in the provided concurrent jobs on clouds environment: (a) the proposed algorithm, (b) the round robin, (c) the throttle, and (d) the leaky bucket. Note that the leaky bucket seems to excel high number of non-conformant data which can affect most multimedia traffic as shown in Table 1. Not to mention, the LB does not balance the load and yet develops very poor throughput figures. Table 2 proves that the RR improves a higher throughput than the LB load balancing algorithm, particularly in the heterogeneous resources (DC1 and DC2). The RR's static scheduler algorithm contemplates the heterogeneous concurrent job along with processing capability of the heterogeneous data centers in order to handle the jobs. So, higher number of jobs gets assigned to the greater capacity on heterogeneous environments then it helps accomplish the job in a tinier time. However, the RR does not perform the load balancing due to the range of traffic load being about 96. Table 3 demonstrates that the dynamic scheduler TH deliberates the load of all its configured data centers and its finishing time of the current load has been accounted. Consequently, it executes in the next level. But if any extensive queues are experienced to the low capacity data centers based on the aforementioned calculation, then it delays the processing end time. The humble RR has not measured any variables about the situation, data center capacities, and the queue lengths. It allocates the concurrent jobs to the data center lists one after another in a systematic manner. Thus, its achievement time of the concurrent jobs is greater than the other two algorithms. Besides, the load range which is 105.6, is yet higher than the RR's one. Finally, the proposed scheduler algorithm computes the estimated queue length in each of the configured data centers and their busyness then the least possible figure has been chosen from the above computations for the arrived concurrent job in one of the data centers and then the concurrent job has been allocated to this data center. Therefore, the proposed algorithm is most seemly to the mixed environment of data centers. If the load balancer finds the data center is extremely loaded from the environment, then that concurrent job is assigned to the least loaded data center rather. The proposed algorithm's results are listed in Table 4. It reveals the load range figure is 4.72 which are the lowest among other algorithms. It proves that the proposed algorithm can balance the traffic load. The results display the proposed algorithm can handle the load balancing while retaining other performance metrics, such as throughput and utilization as well.

Table 1. Load Balancing Simulation Results from LB

Leaky Bucket (LB)					
DC	MQL	THR	MWT	UTL	NonConformance
1	0.39	0.94	0.15	0.29	3.34
2	0.33	0.96	0.09	0.19	2.68
3	4.27	0.48	18.23	0.99	2.72
4	3.09	0.32	11.86	0.82	8.68
Load range	3.94				

Table 2. Load Balancing Simulation Results from RR

Round Robin (RR)				
DC	MQL	THR	MWT	UTL
1	18.52	4.06	63.70	0.99
2	39.47	3.48	177.57	1.0
3	115.13	0.52	248.68	1.0
4	102.08	0.48	231.72	1.0
Load range	96.61			

Table 3. Load Balancing Simulation Results from TH

Throttle (TH)				
DC	MQL	THR	MWT	UTL
1	19.67	3.86	69.43	0.98
2	54.15	3.06	290.24	1.0
3	125.27	0.36	150.34	1.0
4	117.19	0.38	201.56	1.0
Load range	105.6			

Table 4. Load Balancing Simulation Results from PR

Proposed Algorithm (PR)				
DC	MQL	THR	MWT	UTL
1	74.96	3.72	41.83	1.0
2	77.32	3.64	51.64	1.0
3	72.6	0.3	101.8	1.0
4	75.53	0.3	232.2	1.0
Load range	4.72			

5. CONCLUSION

In this paper, the improved load balancing algorithm reflects the capacities of each data center and the job length of each center to allocate the arrived jobs at the balancer into the most suitable center. The proposed dynamic scheduler algorithm studies the load (queue length) of all its configured centers and its uncertain busyness. The performance investigation and simulation results of the proposed algorithm prove that it is most appropriate to the heterogeneous concurrent jobs with mixed resources (data centers) compared to the other round robin, throttle and leaky bucket algorithms. The proposed algorithm regards the QoS and SLA parameters by considering the throughput as a performance metric as well. As part of the future improvements, the balancer processing time can be taken into account. Furthermore, the state of concurrent jobs between the centers in the job migrations will be investigated. The above attentions benefit in further estimation to reduce computation time in the whole algorithms.

REFERENCES

[1] M. G. Pallis, "Cloud Computing: The New Frontier of Internet Computing", *IEEE Journal of Internet Computing*, vol. 14, no. 5, pp. 70-73, 2010.

[2] M. Pearce, S. Zeadally and R. Hunt, "Virtualization: Issues, Security Threats, and Solutions", *ACM Computing Surveys*, vol. 45, no. 2, pp. 1701-1739, 2013.

[3] E. Börger and K. D. Schewe, "Specifying Transaction Control to Serialize Concurrent Program Executions", *Lecture Notes in Computer Science*, vol. 8477, pp. 142-157, Springer 2014.

[4] W. Oortwijn, et al, "An Abstraction Technique for Describing Concurrent Program Behavior", *Proceedings of the 9th International Conference in Verified Software, Theories, Tools, and Experiments,* pp. 191-209, Springer 2017.

[5] Z. Khan, M. Alam and R. A. Haidri, "Effective Load Balance Scheduling Schemes for Heterogeneous Distributed System", *International Journal of Electrical and Computer Engineering (IJECE),* vol. 7, no. 5, pp. 2757-2765, 2017.

[6] C. N. Raghu and A. Manjunatha, "Assessing Effectiveness of Research for Load Shedding in Power System", *International Journal of Electrical and Computer Engineering (IJECE),* vol. 7, no. 6, pp. 3235-3245, 2017.

[7] S. Prabhu, P. Schrammel, M. Srivas, M. Tautschnig, and A. Yeolekar, "Concurrent program verification with invariant-guided under-approximation", *Fifteenth International Symposium on Automated Technology for Verification and Analysis*, pp. 241-248, 2017.

[8] B. Adler, "Load balancing in the cloud: Tools, tips and techniques", http://www.rightscale.com/info center/whitepapers/Load-Balancing-in-the-Cloud.pdf, 2012.

[9] G. Xu, J. Pang and X. Fu, "A load balancing model based on cloud partitioning for the public cloud", *Tsinghua Science and Technology*, vol. 18, no. 1, pp. 34-39, 2013.

[10] K. Nishant, P. Sharma, V. Krishna, C. Gupta, K. P. Singh, N. Nitin, and R. Rastogi, "Load balancing of nodes in cloud using ant colony optimization", *Proceedings of the 14th International Conference on Computer Modeling and Simulation*, pp. 28-30, 2012.

[11] A. Bouajjani, C. Enea and S. K. Lahiri, "Abstract Semantic Diffing of Evolving Concurrent Programs", *Proceedings of International Static Analysis Symposium (SAS),* pp. 46-65, 2017.

[12] C. Sung, S. K. Lahiri, C. Enea and C. Wang, "Datalog-Based Scalable Semantic Diffing of Concurrent Programs", *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18),* pp. 656-666, 2018.

[13] R. Gunawan and K. Mustofa, "Finding Knowledge from Indonesian Traditional Medicine using Semantic Web Rule Language", *International Journal of Electrical and Computer Engineering (IJECE),* vol. 7, no. 6, pp. 3674-3682, 2017.

[14] O. Inverso, T. L. Nguyen, B. Fischer, S. L. Torre and G. Parlato, "A context-bounded model checking tool for multi-threaded c-programs", *Proceedings of the 30th ACM/IEEE International Conference on Automated Software Engineering (ASE '15),* pp. 807–812, 2015.

[15] S. Chaki, A. Gurfinkel and O. Strichman, "Regression verification for multi-threaded programs (with extensions to locks and dynamic thread creation)", *Formal Methods in System Design,* vol. 47, no. 3, pp. 287–301, 2015.

[16] A. R. A. Kumar, S. V. Rao and D. Goswami, "NS3 Simulator for a Study of Data Center Networks", *Proceedings of IEEE 12th International Symposium on Parallel and Distributed Computing,* pp. 224-231, 2013.

[17] R. F. Adler and R. B. Fich, "The Effects of Task Difficulty and Multitasking on Performance", *Interacting with Computers,* vol. 27, no. 4, pp. 430-439, 2015.

[18] P. Marendi´c et al., "An Investigation into the Performance of Reduction Algorithms under Load Imbalance", *Lecture Note in Computer Science,* vol. 7484, pp. 439-450, 2012.

[19] Hong, Yu-Ju & Thottethodi, Mithuna. (2013). Understanding and mitigating the impact of load imbalance in the memory caching tier. Proceedings of the 4th Annual Symposium on Cloud Computing, SoCC 2013. 10.1145/2523616.2525970.

[20] A. Castañeda, S. Rajsbaum and M. Raynal, "Specifying Concurrent Problems: Beyond Linearizability and up to Tasks", *Proceedings of the 29th International Symposium on Distributed Computing,* vol. 9363, pp. 420-435, 2015.

[21] J. Li, X. Lin, S. Nazarian and M. Pedram, "Concurrent task scheduling and storage management for residential energy consumers under dynamic energy pricing", *IET Cyber-Physical Systems: Theory & Applications,* vol. 2, no. 3, pp. 111-117, 2017.

[22] V. Jain and J. S. Prasad, "Solving N-queen Problem Using Genetic Algorithm by Advance Mutation Operator", *International Journal of Electrical and Computer Engineering (IJECE),* vol. 8, no. 6, pp. 4519-4523, 2018.

[23] M. Swarna, S. Ravi and M. Anand, "Leaky Bucket Algorithm for Congestion Control", *International Journal of Applied Engineering Research,* vol. 11, no, 5, pp. 3155-3159, 2016.

[24] S. K. Panda and S. K. Bhoi, "An Effective Round Robin Algorithm using Min-Max Dispersion Measure", *International Journal on Computer Science and Engineering,* vol. 4, no. 1, pp. 45-53, 2012.

[25] Bhagyalakshmi and D. Malhotra, "A Review: Different Improvised Throttled Load Balancing Algorithms in Cloud Computing Environment", *International Journal of Engineering Technology, Management and Applied Sciences,* vol. 5, no. 7, pp. 409-416, 2017.