Indonesian Journal of Electrical Engineering and Computer Science
Vol. 17, No. 2, February 2020, pp. 1109~1116
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v17.i2.pp1109-1116

Cloud computing based load balancing algorithm for erlang concurrent traffic

Chanintorn Jittawiriyanukoon
Graduate School of Advanced Technology Management, Assumption University, Thailand

Article Info

Article history:

Received Jun 2, 2019
Revised Aug 4, 2019
Accepted Aug 18, 2019

Keywords:

Cloud computing
Concurrent programs
Erlang traffic
Load balancing algorithm
Queueing network
Scheduler
Simulation

ABSTRACT

The distribution of scheduler from user inquiries in the clouds is complex. In keeping up with the cloud computing environment and the inquirers, the clouds meet with some problematic load balancing complications as an improving load balancing tool induces the rigorous efficiency of the cloud based website's user access. Overloaded or underloaded conditions originate processing catastrophe regarding the prolonged execution time, bandwidth hog, malfunction, and etc. Besides, to manipulate Erlang concurrent tasks is another skyward situation. Hence, the load balancing is obliged to exhaust all mentioned conditions. The proposed load balancing algorithm for Erlang concurrent tasks (those are and could also be autonomous and unstable.) on VMware workstations is introduced. There are several load patterns within the clouds corresponding to CPU's load (utilization), memory load (queue size), link capacity load (bandwidth), and so on. The proposed load balancing is to spot underloaded and overloaded conditions then stabilizes the weight amidst computing nodes. There are countless load balancing approaches in the cloud environment to examine performance parameters. A short outline of corresponding performance metrics in the review and their findings are presented. To investigate the fit efficiency of the proposed algorithm, the simulation is applied then results based on the proposed method are compared to the existing ones. The outcomes settle the weight balancing, outperform others when executing Erlang traffic, and are catered in the context.

Corresponding Author:

Chanintorn Jittawiriyanukoon,
Graduate School of Advanced Technology Management,
Assumption University, Thailand.
Email: pct2526@yahoo.com

1. INTRODUCTION

Users are looking ahead to more designated applications that come from the swift fetch of precise transactions requested by end-user operation. Virtual machines (VMs) are crucial to a digitally modern organization, and figuring out how they maneuver is the key to the accomplishment of implementing any virtualization solutions. VMs are truly computing power within computing machines. The VM profits tension off the hardware and is associated into single station for the comfortable access and well-being observation. This tolerates IT staff to better audit security issues and allows system administrator to easily maintain [1]. Instead of configuring a firewall for all devices before connecting the network, all data can be secured in the data center. A VM provides user's processes based on the abstraction for virtualization [2]. It is the level where the VMs opt for the physical part the common type of virtualization occurs. The physical part operates and controls VMs on a centralized server (on-premise host or cloud based host) in order that it can be easier to reach by several users. In practice, we can carry out several VMs on a host machine to opt PC virtualization technology. Data is not warehoused on the user devices any longer, which only need unintelligent devices to do online access. The VM per se can reside on the device as well to access the same

information in the clouds. Moreover, VMs can support organizations a security issue. As user devices are kept on the host machine, tolerating IT staff to secure all users at once. This level of security cuts off turnaround time when the attack starts; possibly securing associated transactions. The VM protects by allowing users to steadily access records distantly and rejects security threats from unauthorized devices in the network [3].

Metzler, et al [4] present that the noticeable benefit of software test is a harsh inspection of software behaviors. In the case of bulky state spaces of concurrent software and probabilistic scheduler, the test process is slow as well as leads to a huge delay between the utilization and completion of concurrent programs. They outline an innovative approach to verify concurrent software and to reduce the latency. By controlling the execution of concurrent software due to a trivial set of tolerable schedules, verification time is low. The evaluation based on benchmark software confirms that the overhead due to execution time is practical to existing theoretical frameworks. Another article treats a memory for concurrent software by restricting the abstraction level [5]. *Hanxleden, et al* [6] mention that the synchronous language confirms fixed concurrency but the cost of restrictions on concurrent programs is considered whether it is constructive or valid. The serial C language and Java script provide an intuitive programming model but give no guarantee to fixed concurrent programs. They investigate classes of shared accesses, define serial set of tolerable scheduling as a picture of "*free scheduler*", then stem the concept of serial productiveness, and apply a priority to the scheduling algorithm for evaluating software efficiency. *Liang and Feng* [7] describe an existing research on verifying concurrent tasks only focuses on certainty, e.g., linearization or correctness. These things are challenging to verify as they tolerate the progress of a single thread to another based on the unweighted scheduler. The authors propose new program logic with guarantee for testing linearization then progressing concurrent tasks under fair scheduler. Finally, they apply the program logic to verify deadlock-free algorithm for instance queue locks or coupling lock.

The verification of concurrent applications written in low-level languages is introduced by [8]. The verification solves the state-space explosion problem due to multiple threads execution. Bounded Model Checking (BMC) cannot iron out the problem as such in practice. In this research, the authors propose a method to conquer the constraint of the state-space used in the BMC technique. They apply under-estimations of lazy demand-driven improvement and data flow of shared memory. They also have implemented a prototype to evaluate potential performance metrics. Articles on load balancing in the cloud computing environment are outlined in [9, 10].

To distribute the workload on the distributed processors is important. It is supporting divides and conquers procedure, which is the cost-effective scheme for load balancing complications. Load balancing becomes the main issue for scheduler in Cloud computing system. The authors [11] propose algorithm based on optimization techniques such as min-max and max-max schemes and apply for FCC network. Performance metrics such as speedup and utilization are collected. Results based on the proposed algorithm show better in comparison to existing work under several conditions. Nevertheless, the load balancing algorithm in clouds is a subject which needs novel schemes to deal with several modifications. Further, there are load balancing schemes, such as Throttle, Leaky bucket, and Round Robin [12] in order to maximize the proficiency of cloud computing system.

Concurrent software with particular structures such as probabilistic behavior and synchronization makes the verification intricate. *Sahoo and Ray* [13] find several kinds of concurrent weaknesses. The authors apply a mechanism called Symbolic Path Finder (SPF) which is an improvement of Java Path Finder (JPF) for concurrency verification. SPF checks concurrent weaknesses such as race condition, deadlock, etc. The proposed SPF also engenders execution tree of the source code for verification case. The verification points out the type of defects in the given code. *Sergey, et al* [14] propose a test tool based on Fine-grained Concurrent Separation Logic (FCSL). It is a program-logic for testing software characteristics such as threads spawning and higher-order functions. By considering a deterministic concurrent program, based on restricted commutation and the state space, FCSL can test a number of concurrent software through proofs about libraries in a local thread, reuse and scalability. A framework for stationary synthesis of fixed concurrency control of the parallel system is presented in [15]. The algorithm familiarizes synchronization that converts the software into a deterministic one by calculating a local thread order to assure determinism and preserve the termination. The synthesis algorithm based on two abstraction levels, memory and thread abstractions, records the array access. *Hawkins, et al* [16] explain a method for synthesizing concurrent data representations. The compiler takes the input software with concurrency and synthesizes a representation of the threads as sets of data structures and an attainment of locks to coordinate concurrent level. The results support the deadlock-free operation.

The focus of the paper is to consider the efficiency of load balancer with Erlang concurrent programs using a simulation [17]. In the research, features of the Erlang concurrent traffic is deliberated in the Section 3. In order to leverage the Erlang concurrent traffic, the proposed load balancing algorithm is

presented then results are validated in terms of the correctness and effectiveness by comparing to results from other algorithms. Furthermore, other performance metrices are displayed.

2. LITERATURES REVIEW

Single thread is a simple part of CPU's utilization. Multithread is a processing pattern that tolerates a process to run various code fragments (threads) concurrently within the "environment" of the process. A thread like the media player starts its one thread for handling a clip while another thread for uploading a new media for a different user. Multithread manages its execution by several independent users, or various requests from a single user by avoiding the software duplication. The up-to-date OS such as Microsoft Windows manages multitasking environment through concurrent process [18]. In multitasking, the processing is the concurrent execution. New task repeats prior to the finishing process of others, without awaiting others to complete. The multitasking takes parallel processing simultaneously, rather than opting sequential fashion. Multitasking is also a common feature of the up-to-date operating systems. It takes a benefit of computer resources; while software waits for I/O cycle to complete, the CPU avails for other processes of the same. In a time-sharing scheme, distinct processes run on a single CPU while the only processor is offering users by multitasking through multiple threads.

However, the unstable concurrency results a performance pressure on load balancing challenges. Dynamic load balancing algorithm is outlined to lower the response time for virtual machines in cloud systems [19]. Not to mention Erlang distributed traffic roots lengthy problems such as load imbalance and linearizability. Load imbalance for real-time traffic is characterized and demonstrated in [20]. Furthermore, concurrency also creates prolonged scheduling problems in the distributed network [21]. Hence, the study in this paper involves with distributed system of unstable Erlang concurrent traffic, especially presents how to shape Erlang concurrent tasks then balance the load regarding storage control. The proposed algorithm manages the Erlang traffic to fit the capacity of individual computing center. The previous three algorithms such as the Least Utilization (LU), the Cyclic Manner (CM) and the Least Load (LL) are benchmarked for the validation of the proposed algorithm. Experimental results based on simulation tool are summarized to inspect the effectiveness of the proposed algorithm. The remaining of the paper is organized as follows. In next section, a review about the characteristic of Erlang traffic is discussed. Section 4 explains how to balance the load based on previous algorithm and the proposed algorithm. Section 5 displays the summary of simulation results and the subsequent section includes the conclusion.

3. ERLANG CONCURRENT TRAFFIC

The reason for centering on Erlang function instead of others is Erlang's capacity to leverage distributed programming and concurrency. Concurrency means programs that can accommodate various threads of execution simultaneously. For instance, modern operating systems offer the user to run a spreadsheet, a word processor, a mail client, and other jobs all executing at the same time. Each CPU in the computing environment is controlling one job (or thread) at a time, but it switches between the threads at rate that it provides the impression of executing them all simultaneously. It is practical to design parallel processing in an Erlang fashion and to tolerate these jobs to communicate with each other. In Erlang distribution, the thread of execution refers to a process. The "*process*" is called when the threads of execution are independent with each other. The key determination is to schedule concurrent tasks to the VMs according to flexible time which involves in discovering a correct sequence where concurrent tasks are manipulated under program logic constraints [22]. The scheduler of cloud computing is a threat. To keep up with the threat the number of effective scheduling algorithms is reviewed. It targets at a scheduler for handing over users' concurrent tasks. The greedy algorithm is presented for the VM data migration from a data center to others. VMs within a cluster are set with priority which lessens the migration time. Then it is simulated and results point out that the link bandwidth is significant parameter in the migration [23].

This section defines the experimental feature of the unstable current software. The primary application software enters the "*split*" node for a number of isolated concurrency. The arrival time follows Erlang distribution. The computing power includes so called parallel processors. The primary software enters the "*split*" node at no latency, but will be partitioned into a number of specified concurrency. The fragmented software from the corresponding primary one are called *relatives*. All relatives attend parallel processors at once after visiting the balancer. It is entirely isolated relatives, excluding the queue's effect while waiting for parallel services. The service time of all fragmented ones follows the exponential distribution. If a relative completes the execution, it emerges the "*join*" node to wait until reuniting for all relatives as shown in Figure 1. After all relatives' reunion, they once fuse into the primary software and stamp the synchronization to proceed to another round. The time interval from appearing at the "*split*" node to the completion of fusion

refers to a cycle or the expectancy. If the software (S_m) is fragmented in the "*split*" into s_{mn} relatives then it develops the concurrency (s_{mn}) throughout the cycle. The concurrency s_{mn} alters in each cycle after the fusion. The matrix of any software (S_m) contains a row of cyclical concurrency figure is denoted in (1). The S_m matrix is a deterministic set. A component s_{ij} is set of the unstable concurrency whenever $\{s_{ij} \geq 0,\ 1 \leq i \leq m;\ 1 \leq j \leq n\}$. An example of matrix for two software (S_1 and S_2), where S_1=[1 2 1] and S_2=[0 4 0] is shown in Figure 2.

$$\begin{bmatrix} S_1 \\ S_2 \\ M \\ S_m \end{bmatrix} = \begin{bmatrix} s_{11}\ s_{12}\ \Lambda\ s_{1n} \\ s_{21}\ s_{22}\ \Lambda\ s_{2n} \\ M\ M\ O\ M \\ s_{m1}\ s_{m2}\ \Lambda\ s_{mn} \end{bmatrix} \quad (1)$$

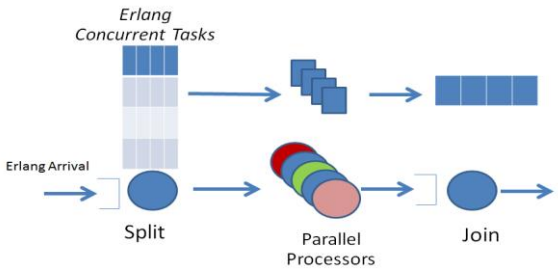

Figure 1. The experimental model

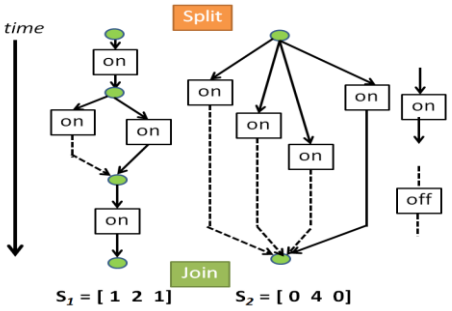

Figure 2. Example of unstable software

4. CLASSIFICATION OF LOAD BALANCING ALGORITHMS

The overloading of the computing nodes is not only occurring to data centers, but it also arises to the corporate servers which have a substantial number of users. It is thus a gigantic problem for huge organization and many corporate. However, there is a solution for such problem which is defined as a fancy word "*load balancing*". Load balancer is an excellent mechanism applied for improving the total capacity of the computing node by allocating the loads across the several servers. In general, databases, software access, web traffic, and other things, which exhibit the weighty loads opt the load balancer application to handle the users with no disruption. In case of the single server for everything, the computing machine ends up in sluggish processing of the users request or becomes an unable response, which is not acceptable in the real-life process of the e-commerce site. Although, in regular days the servers are adequate to serve all the user requests, but sometimes the traffic goes beyond the prediction. It thus results that servers battle to react all queries, and users experience delayed loading matter. Finally, the server cannot take any longer user requests for the services which affect corporate revenue and sales. Not to mention it strikes the organization reputation among the users. In this case a load balancer for web servers will be the only solution. To diminish this sort of complications, the load balancing approach on the cloud network is significant. To apply load balancer, at least one or more attached to servers can resolve which server it allocates the requests for further process.

4.1. Least Utilization (LU)

The load balancer alleviates the user requests in the fashion that it shapes the traffic load. The LU algorithm [24] sets a selection criterion for the data center with the largest remaining unused capacity and shapes the load into a dispatch of packets in the cloud network. In this algorithm the load balancer allocates queries for the data center it scans fully until the data center with the lowest utilization factor is found. The LU is also to manage the metered-usage of any centers to avoid going beyond the limit and to prevent an extra charge.

4.2. Cyclic Manner (CM)

It is the humblest algorithm that makes use of the notion of time slices. Here the time is distributed into several slices, each node representing the data centers is specified a particular time interval and the node

executes its tasks. The cloud network resources are given to the users based on this time slot. In cyclic manner [25], if the time slot is large enough then the algorithm is exactly identical to the first-in-first-out discipline. It is the commonly used and fairest algorithm, as it is simple to apply and there is no priority concern, only an FIFO scheduling and a time interval for the resources.

4.3. Least Load (LL)

The LL algorithm [26] is needed algorithm for the load balancer. The algorithm directs cloud network traffic to the data center with the least number of active loads. The load balancer always keeps observing the loads. The LL algorithm dispatches all tasks across the cloud network and guarantees that no computing nodes are underloaded, overloaded or idle. It is one of the dynamic scheduling algorithms as it needs to count the amount of current loads for all computing nodes. The load balancer observes the load of each data center, decreases the figure when a task is taken out of the buffer, and increases the number as the new task arrives.

4.4. Proposed Algorithm (PR)

The proposed algorithm executes the designated unstable software which is spawned into S_{mn} concurrent tasks in the "*split*" node. The load and utilization are monitored to confirm tasks dispatch in each and every computation steps. To find the best-fit sub-solution (*local*) in each step, the proposed algorithm looks into the sub-solution results from that step. The decision is then finalized using the narrowest gap between the S_{mn} and the sub-solution. This reveals an overall (*global*) result to accomplish the best-fit consequence of a goal. It is a mechanism of picturing a single step ahead. The proposed algorithm obliges straightforward rule when it computes all sub-solutions at each fractional step which leads the cost of computation to $O(mn)$. The proposed algorithm is summarized in Figure 3.

Proposed Algorithm (PR)

Require: Concurrency matrix $[S]$ with m rows and n columns
Ensure: A=all candidates in each computation step=$\{A_1^t, \ldots, A_y^t\}$,
B_y^t=load amount in step y, U_y^t=utilization factor in step y,
O_y^t=specific candidates in step y, P_r=a premium solution where
$P_r(A_y) \geq 0$ and $A_y \in A^t$

for J=1 to n do
$O_y^t \leftarrow 0$
for k=1 to y /** All candidate seats **/
$A_k = w_1 * U_k^t + w_2 * B_k^t$
/** w_1 and w_2 are two weighted values used to find out the load figure and utilization **/
F=arg $max_{A_y \in A^t} P_r(A_k)$
/** Solution F corresponding to U_y and B_y **/
end for

for r=1 to m
/** Select the concurrent task to fit the computing node **/
$\Delta_r = F - S_{rj}$
If $\Delta_r \geq 0$ **then**
O_y^t=arg $min_{A_y \in A^t}(\Delta_1, \Delta_2, \Delta_3, \ldots, \Delta_m)$
** Update new best for this step **/
end if
end for
Return O_y^t
end for

Figure 3. Proposed algorithm

Figure 4 depicts a typical load balancing design used in cloud computing system where the balancer stabilizes the load following these classical four phases: a) Collects incoming inquiries (Erlang traffic) from multiple users b) The "*split*" node spawns incoming requests from users into various concurrent tasks and forms a queue c) Checks the current load status of the computing nodes (data centers) in the cloud from time to time using a daemon software and d) Employs a balancer algorithm to choose proper node.

The complicated computing systems route millions of packets in a second. The huge traffic as such must be dispatched proficiently among the available computing nodes in order that the nodes can leverage them efficiently without any impacts on users. Hence, key advantages of the balancer are helping in managing and controlling Erlang traffic, raising the computing node's utilization, balancing the load based on node capacity, improving the availability and lowering over-provision of the network infrastructure. The aim of cloud architectures is to provide speedy elasticity then whatever the application is clouds have to meet on-demand services of the user.

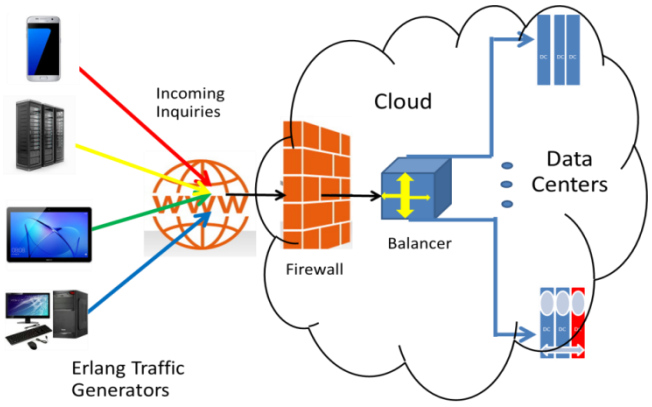

Figure 4. Proposed architecture

5. RESULTS AND ANALYSIS

In cloud environment, concurrent tasks designate the resources on data centers. If a task occupies the resource at any specified time, other tasks must wait in queue. Systems as such can be modeled with the queuing network. Queuing concept helps shape the time that tasks expend in various queue networks [27]. The open queuing network allows software to enter the environment at a "*split*" and leave at a "*join*" as shown in Figure 5. These "*split*" and balancer nodes take no service time. Queue discipline for all nodes is identical to FIFO basis. Servers 1 to 4 represent data center nodes with equal branching probability. The program-dependent service time functions for the data centers 1 and 2 follow the exponential distribution with mean of 1, 2, 3, and 4 seconds for four software (S_1,S_2, S_3, and S_4) respectively. They detect exponential function with mean of 2, 4, 6, and 8 seconds for the data centers 3 and 4. The software arrival rate at the "*split*" expects Erlang-2 function with mean of two in a second.

Erlang concurrent load from four sources for balancing experiment is specified in (2). The simulation [17] runs and shows analytical parameters (such as mean queue length (MQL), throughput (THR), mean waiting time in queue (MWT), utilization (UTL), etc.).

$$\begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} 2\ 2\ 6\ 7\ 8 \\ 0\ 2\ 1\ 2\ 0 \\ 7\ 2\ 6\ 4\ 1 \\ 0\ 1\ 0\ 6\ 8 \end{bmatrix} \quad (2)$$

The simulation results are summarized in Table 1 to 4. The performance metrics of these 4 algorithms with Erlang concurrent software on the cloud environment: (a) the least utilization, (b) the cyclic manner, (c) the least load, and (d) the proposed algorithms are enumerated in these tables. Erlang concurrent software is designed to meet the resource capacity on cloud environments and to help achieve the software speed-up. However, these LU, CM, and LL algorithms do not balance the load as the range figures of Erlang traffic load are 60.7, 72.26, and 58.95 as depicted in Table 1, 2, and 3, respectively. The proposed algorithm shown in Table 4 gives the lowest figure of load range (=14.94) and balances the load better than other algorithms, particularly in the heterogeneous resources (four data centers). It is because traditional algorithms (static scheduler algorithms) have not taken any considerations about the current situation, data center capacities, and the queue lengths. It allocates the concurrent jobs to the data center lists one after another in a simple manner.

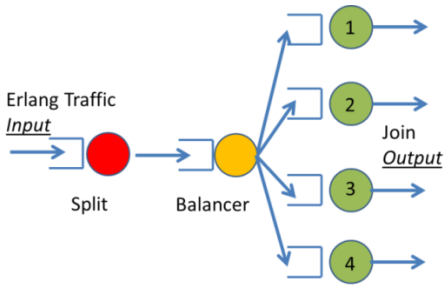

Figure 5. Simulation model

Table 1. Load Balancing Simulation Results based on LU

Least Utilization (LU)				
Server	MQL	THR	MWT	UTL
1	43.1	0.42	67	1.0
2	19.9	0.38	32.8	0.97
3	80.6	0.17	59.8	1.0
4	76.5	0.21	58.9	1.0
Load range	60.7			

Table 2. Load Balancing Simulation Results based on CM

Cyclic Manner (CM)				
Server	MQL	THR	MWT	UTL
1	10.6	0.4	20.3	0.99
2	25.42	0.4	40.2	0.99
3	77	0.19	54.5	1.0
4	82.86	0.21	64.18	1.0
Load range	72.26			

Table 3. Load Balancing Simulation Results based on LL

Least Load (LL)				
Server	MQL	THR	MWT	UTL
1	33.05	0.38	52.65	1.0
2	39	0.33	61.16	1.0
3	92	0.14	38.45	1.0
4	63.6	0.2	45.38	1.0
Load range	58.95			

Table 4. Load Balancing Simulation Results based on PR

Proposed Algorithm (PR)				
Server	MQL	THR	MWT	UTL
1	41.16	0.37	62.1	1.0
2	45.02	0.26	55.1	0.98
3	56.1	0.21	65.4	1.0
4	42.58	0.26	50.13	0.98
Load range	14.94			

6. CONCLUSION

In this paper, a new mechanism of balancing load has been approached by using dynamic parameters like utilization and present load amount. From the simulation results investigation the proposed load balancing algorithm is apparently the most efficient technique than LU, CM, and LL algorithms. The three algorithms belong to the cloud environment and are primarily regarded to static balance whereas the proposed algorithm is distributed balancing algorithm. As the load range plays important role in cloud computing systems, then the proposed algorithm resembles as the finest among the other traditional three algorithms. Future research centers on devising the proposed algorithm by tallying additional parameters.

REFERENCES

[1] F. Abazari and M. Analoui, "*Exploring the effects of virtual machine placement on the transmission of infections in cloud*", The 7th International Symposium on Telecommunications, pp. 278-282, 2014.
[2] B. Kruck, T. Pape, T. Felgentreff, and R. Hirschfeld, "*Crossing Abstraction Barriers When Debugging in Dynamic Languages*", Proceedings of the Symposium on Applied Computing, pp. 1498-1504, 2017.
[3] T. Mehraj, M. A. Sheheryar, S. A. Lone, and A. H. Mir, "A critical insight into the identity authentication systems on smartphones", *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 13, no. 3, pp. 982-989, 2019.
[4] P. Metzler, H. Saissi, P. Bokor, and N. Suri, "*Quick verification of concurrent programs by iteratively relaxed scheduling*", Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, pp. 776-781, 2017.
[5] 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.
[6] R. V. Hanxleden, et al, "Sequentially constructive concurrency - A conservative extension of the synchronous model of computation", *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 4, article. 144, pp. 144:1-144:26, 2014.
[7] H. Liang, and X. Feng, "*A program logic for concurrent objects under fair scheduling*", Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 385-399, 2016.

[8] 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.
[9] K. Ramana, and M. Ponnavaiko, "AWSQ: an approximated web server queuing algorithm for heterogeneous web server cluster", *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 3, pp. 2083-2093, 2019.
[10] K. S. Qaddoum, N. N. E. Emam, and A. Mosleh, "Elastic neural network method for load prediction in cloud computing grid", *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 2, pp. 1201-1208, 2019.
[11] 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.
[12] C. Jittawiriyanukoon, "Performance Evaluation of Proposed Load Balancing Algorithm with Unstable Concurrent Programs", *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 14, no. 3, pp. 1452-1459, 2019.
[13] B. K. Sahoo and M. Ray, "Concurrency testing using symbolic path finder", *International Journal of Engineering and Technology*, vol. 7, no. 2.6, pp. 275-282, 2018.
[14] I. Sergey, A. Nanevski, and A. Banerjee, *"Mechanized verification of fine-grained concurrent programs",* Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 77-87, 2015.
[15] V. Raychev, M. Vechev, and E. Yahav, "*Automatic Synthesis of Deterministic Concurrency*", International Static Analysis Symposium, pp. 283-303, 2013.
[16] P. Hawkins, A. Aiken, K. Fisher, M. Rinard, and M. Sagiv, "*Concurrent data representation synthesis",* Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 417-428, 2012.
[17] http://www.bkhoshnevis.com/
[18] G. Teo, L. R. Jones, and G. Matthews, "*Predicting Task-Induced State Changes in a Multitasking Environment from Personality Factors*", Proceedings of the Human Factors and Ergonomics Society Annual Meeting, vol. 62, no.1, pp. 762–766, 2018.
[19] S. Garg, D. D. V. Gupta and R. K. Dwivedi, "*Enhanced Active Monitoring Load Balancing algorithm for Virtual Machines in cloud computing*", International Conference System Modeling & Advancement in Research Trends (SMART), pp. 339-344, 2016.
[20] Q. Huang, et al, "*Characterizing Load Imbalance in Real-World Networked Caches*", Proceedings of the 13th ACM Workshop on Hot Topics in Networks, pp.1-8, 2014.
[21] O. M. Elzeki, M. Z. Rashad, and M. A. Elsoud, "Overview of Scheduling Tasks in Distributed Computing Systems", *International Journal of Soft Computing and Engineering*, vol. 2, issue 3, pp.470-475, 2012.
[22] A. Lakra and K. D. Yadav, "*Multi-Objective Tasks Scheduling Algorithm for Cloud Computing Throughput Optimization*", Procedia Computer Science, vol. 48, pp.107-113, 2015.
[23] S. S. Baghshahi, S. Jabbehdari, and S, Adabi, "Virtual Machines Migration based on Greedy Algorithm in Cloud Computing", *International Journal of Computer Applications*, vol. 96, pp. 32-35, 2014.
[24] S. Singh, S. Tripathi and S. Batabyal, "*Utilization Based Secured Dynamic Scheduling Algorithm for Real-Time Applications on Grid (U-SDSA)*", IEEE International Conference on Advanced Information Networking and Applications (AINA), pp. 606-613, 2017.
[25] D. Patel and A. S. Rajawat, "Efficient Throttled Load Balancing Algorithm in Cloud Environment", *International Journal of Modern Trends in Engineering and Research*, vol. 2, issue 3, pp. 463-480, 2015.
[26] M. E. Mustafa, "Load Balancing Algorithms Round-Robin (RR), Least Connection, and Least Loaded Efficiency", *GESJ: Computer Science and Telecommunications*, vol. 51, no. 1, pp. 25-29, 2017.
[27] J. Troya and A. Vallecillo, "Specification and simulation of queuing network models using Domain-Specific Languages", *Computer Standards and Interfaces*, vol. 36, pp. 863–879, 2014.