TELKOMNIKA Telecommunication, Computing, Electronics and Control
Vol. 18, No. 4, August 2020, pp. 1795~1801
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v18i4.13150

Evaluation of load balancing approaches for Erlang concurrent application in cloud systems

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

Article Info

Article history:

Received May 18, 2019
Revised Apr 1, 2020
Accepted Apr 13, 2020

Keywords:

Cloud virtualization
Erlang distribution function
Load balancing
Queueing network analysis
Simulation

ABSTRACT

Cloud system accommodates the computing environment including PaaS (platform as a service), SaaS (software as a service), and IaaS (infrastructure as service) that enables the services of cloud systems. Cloud system allows multiple users to employ computing services through browsers, which reflects an alternative service model that alters the local computing workload to a distant site. Cloud virtualization is another characteristic of the clouds that deliver virtual computing services and imitate the functionality of physical computing resources. It refers to an elastic load balancing management that provides the flexible model of on-demand services. The virtualization allows organizations to improve high levels of reliability, accessibility, and scalability by having a capability to execute applications on multiple resources simultaneously. In this paper we use a queuing model to consider a flexible load balancing and evaluate performance metrics such as mean queue length, throughput, mean waiting time, utilization, and mean traversal time. The model is aware of the arrival of concurrent applications with an Erlang distribution. Simulation results regarding performance metrics are investigated. Results point out that in Cloud systems both the fairness and load balancing are to be significantly considered.

Corresponding Author:

Chanintorn Jittawiriyanukoon,
Graduate School of Advanced Technology Management,
Assumption University,
88 Moo 8, Bang Na Trad Km 26, Bang Sao Thong District, Samut Prakan Province 10570, Thailand.
Email: pct2526@yahoo.com

1. INTRODUCTION

The cloud system load balancing services deliver load distribution from one source to various destinations accessible from cloud virtualization network (CVN). The services provide a traffic balancer with an option of a private or public IP address, and attainable speed. The traffic balancer helps improve the efficiency of resource utilization, scalability, and ensure better accessibility. We can design load balancing approaches and application-centric policies to guarantee that the traffic balancer distributes load only to vigorous destination nodes. The traffic balancer helps lessen the window size by spilling load from a feeble node prior to discard them from the system as non-conformance [1]. Normally the traffic balancers are in between the client and the server nodes checking incoming application traffic and allocating the load across many destinations based on load balancing algorithms. By balancing incoming applications across the clouds, a traffic balancer lessens specific traffic and avoids a single point of failure, therefore keeps improving overall application responsiveness and availability [2]. The research study [3] uses the graphic CPUs for the traffic calculation and refers to pluses and minuses amount. However, it is not concrete if the traverse time is not relating the overhead of splitting and merging, especially in terms of concurrent applications. In this regard, the experimental model proposed in the research aims at taking both splitting and merging time into account. This paper is organized as follows. First, section two outlines related works on a cloud virtualization and the processing features. Section three presents the queueing model approximation while section four discusses about load balancing approaches. Section five demonstrates simulation outputs and analysis. The last section encapsulates conclusion and some scopes of future investigation.

Clouds use virtualization technology to lower physical costs and fasten the distribution. Clouds virtualization partitions a physical space on servers into many virtual machines (VMs) with the independent computing environment. Each VM runs different OSs and applications; the virtualization cuts the need for additional hardware infrastructure and affords a cost-effective way to run multiple services. Malhotra et al. [4] discuss the virtualization in Cloud systems [5-11]. Authors start with an introduction, characterization and history of cloud systems. Architecture of virtualized technology has also been discussed and comprehensively described. Besides, fundamental concepts like virtual servers, categories of virtualization, and advantages of virtualization technologies are also deliberated. In conclusion, future scopes of virtualization and cloud systems are outlined. Authors have thoroughly discussed diverse challenges to cloud systems [12, 13] such as infected applications, noisy data, and data integrity.

Xing et al. [14] explain that Cloud systems are the basic conversion in the area of IS. It is a depiction of an association towards the large scale and thorough knowledge. Virtualization is the critical characteristic of cloud systems. To use virtualization provides not only the efficiency, but also big experiments in the area of information security and private data. It ties different virtual physical resources to the same hardware, thus other users data is accessible. The paper analyzes multiple mechanisms to improve and solve the problem of the cloud security. Velde and Rama [15] present the swift development of cloud systems which offer excessive convenience to developers. They explain that the resource management of the cloud hardware has become a popular topic, specifically the load balancing approaches in data center are the key to cloud services provider. Authors propose the load balancing framework for hardware by utilizing the window size strategy and the prediction model to cut the cost of the data migration. Authors also propose that the algorithm in this paper can successfully reach load balancing, uphold the utilization of the physical VMs, and solve the data migration problem.

Venkatachalapathy et al. [16] outline that the presence of big data is a stimulating and cloud computing system has to deal with handling the large amount of data at the remote server with full of clouds integrity. Pattern recognition (PR) algorithm is applied to avoid redundancy during the development of the dataset. A dataset is patterned whenever data is uploaded. Authors define a cloud system in which the dataset is split based on balanced splitting method. The medical applications on the database are used and promote virtualization in the physical platform for resources management. VMs can be discarded or augmented abased upon the request. The results demonstrate that cloud systems are improved in terms of virtual environment setting.

Queueing experimental model enables the investigation with the relations of a complex cloud system. Informational and traffic flows can be simulated and the alterations to the model's behavior can be analyzed. Results obtained from analyzing the model can be significant and improve the system. Switching simulation input parameters helps understand how specific factors of the system correlate. Simulation [17] is commonly used to investigate new designs or verify analytical solutions before the implementation starts and cut down huge cost. Experimental model based on queueing theory is used to model the flow of applications traffic in cloud systems. This paper introduces an overview of how to model cloud networks using the queueing experimental model. In practice traffic flows through multiple nodes, the extension of traffic analysis towards network model is essential. In this paper, a cloud network can be simulated as a queueing network model where traffic generated by applications consumes time at each node. The spent time is independent among each concurrent application upon the occupation of the network nodes, i.e. a high consumption leads to en route congestion and load imbalance. Accordingly, the performance metrics of the simulation model are used to figure the traverse time on the cloud.

The model represents several processing nodes in which application can spend time in taking a service. First of all, the concurrent application is partitioned into *n* independent tasks. Tasks which meet the busy stage of each node must queue up for their turn at queue's buffer dedicated to the service node. These buffers and the servers develop the queuing model. There are numerous models which can form a network of queue, such as communication networks, transportation systems, road networks and etc. First, the queueing model can be regarded as the task's arrival time is described by a distribution function. In practice, an application involves an Erlang distribution. Erlang traffic works well on cloud platforms as real-time applications are involved. In designing video stream over cloud-based architecture, Erlang traffic model is generally used. So, Erlang semantics, which represent concurrent, distributed and HotSwap loadable applications could be assumed for real-time and multimedia applications. Erlang traffic for multimedia can be analyzed in [18]. Authors concern of Erlang distribution for practical case with video stream flowing in the Internet network. Erlang formulas are simulated, and verification is defined by using the designated video stream flowing in the Internet network. Additionally, SNMP protocol in the same paper approves the information base for Erlang traffic from distributed nodes. Applications over Internet network which is not implemented for treating multimedia traffic can be expressed in Erlang function. Traffics like video stream transmission are taken into consideration and investigated to guarantee the quality of service (QoS). The QoS problem is fixed by using the queuing model with Erlang equation which is assumed for latency, required bandwidth, and availability calculation.

Second, the discipline of queueing tasks is waiting in the buffer or perhaps giving up afterwards. Telephone operators may experience the hang-up calls from customers as they cannot be patient until a succeeding assistant avail. Third, it is the service time which is arbitrary but shadows an exponential distribution. The busy period in each node is due to the queue length in the buffer. Analogously, the processing time of each customer at the call service is heaped up if the number of waiting calls is increasing. Fourth, the service distribution reveals what service pattern each task requires, one on one or so. Service patterns can be forecasted based on their arrival distributions, such as last come first serve, priorities, first come first serve, or resource-sharing. Fifth, the service power is a single unit or multiple units to treat the tasks. Last, the queue's number is the number of waiting tasks in the buffer. In the computer network, limited packets are acceptable at a buffer. The storage space is a key value in the network design.

The experimental queueing model is illustrated in Figure 1. First, the Erlang applications are split into *n* concurrent tasks. The processing time at the splitter depends directly on the concurrent number. We assume that *n* units of processors facilitate these concurrent tasks in the parallel fashion. Applications arrive the system with Erlang distribution, attend the splitter first to build *n* tasks. These *n* tasks are called *families*. The whole family members proceed instantly to *n* processors.

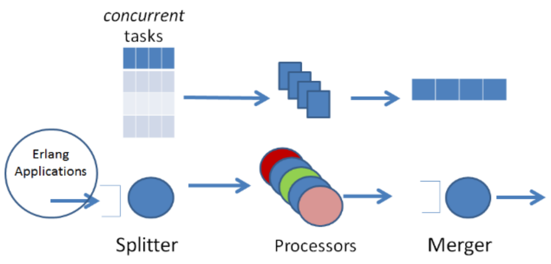

Figure 1. *Ek*/*M*/*n* model

Although tasks are independently executed but priority tasks decline the traverse time in the system as presented in [19]. These tasks may worsen the waiting time in merging somehow. When task's service is finished from the processing environment, they transport to the merger immediately and keep waiting in the buffer for all members' completion before reassembling into their original application. At this moment, the merging execution time is added. The traverse time which begins by arriving in the splitter till completing at the merger is called turnaround time (T_{TT}) as listed in (1). Simulation [20] is contributed for collecting the performance metrics such as T_{TT}, throughput and utilization factor for the analysis. The *Ek/M/n* simulation model depicted in Figure 2 is investigated. Erlang applications enter *the splitter* with k/λ mean arrival time and a concurrent task takes the mean $1/\mu$ service time. The application's splitting time ($T_{Splitter}$) based on the concurrent number is added then the merging time (T_{Merger}) based on the same is collected whenever the completion of the whole members is achieved. The Δ_c is the service time taken at the n^{th} processor of the n^{th} task.

$$T_{TT} = T_{Splitter} + \arg\ \max_{c\ \in\ n}(\Delta_1, \Delta_2, ..., \Delta_c) + T_{Merger} \qquad (1)$$

2. LOAD BALANCING APPROACHES

Concurrent task's scheduler [21] is vital due to the load balancing issue on parallel processing system. It is inciting on processing powers whenever concurrent applications are investigated. Especially, unstable concurrent applications, that is, an application made of a combination of separate but alike tasks, are serviced on a parallel environment. In addition, these concurrent applications are developing complicated problems, including computation overhead, load imbalance, and multiple threads on multi-core VMs.

If the processors are alike for all transport bandwidths, then a leaky bucket algorithm [22] can improve the throughput. However, if dissimilar processors are connected with changeable bandwidths then this algorithm dissatisfies since it is crucial to transfer tasks to the appropriate processor.

The following section describes the feature of current applications referred in the experimental model. First, applications exhibit *n* independent concurrency. We assume that the processing power also comprises *n* parallel nodes accordingly. After splitting, the *n* concurrent tasks so called *families* enter the *balancer* in the cloud at no delay time prior to the emergence of parallel processors (a representation of data center in the cloud computing system). It is absolutely separate amongst *members*, but the impact of the queue length. The service time follows the exponential function. If a *member* completes the service at data center, it enters the *merger* to pause for the completion of all *families* as shown in Figure 2. If all members are done, they are assembled into the application. It scripts the synchronization of the *family* and moves on to next cycle. If an application comprises *n* concurrent tasks (A_{nm}) in the m^{th} cycle then it is partitioned into A_n tasks in the *splitter*. Note that A_{nm} repeats and changes cyclically after family members are merging. The concurrency matrix of applications contains a row of mutable concurrency. The A_n matrix is anticipated to be a deterministic set. A component A_{ij} is set for the mutable concurrency whenever $\{A_{ij} \geq 0,\ 1 \leq i \leq n;\ 1 \leq j \leq m\}$. A sample of concurrency for two applications (A_1 and A_2), where $A_1 = [1\ \ 1\ \ 2\ \ 3]$ and $A_2 = [1\ \ 2\ \ 0\ \ 0]$ is shown in Figure 3.

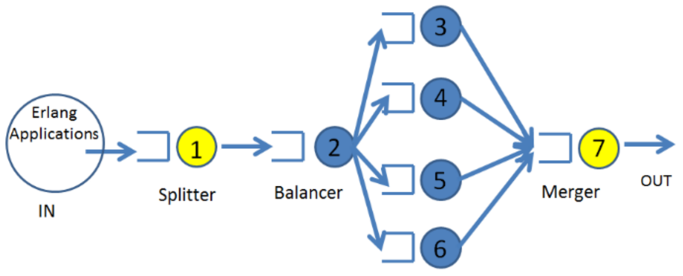

Figure 2. Simulation model

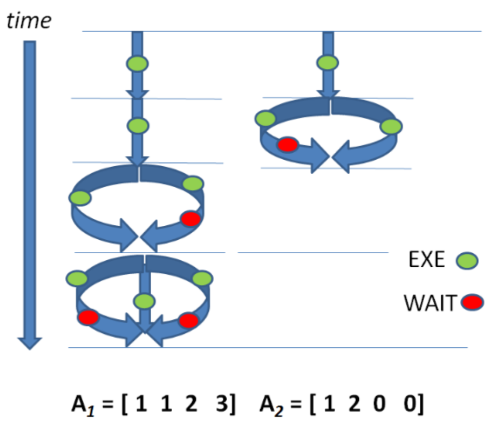

Figure 3. Sample of unstable concurrent applications

With the emergence of cloud systems, the impression of balancer between the cloud systems, VMs, data centers and cloud nodes has grown the attraction. Many research studies are increasingly concentrating on to improve the performance of the balancer at ultimate load period. Load balancer is important machine equipped in the hybrid clouds to fulfill the scope of the quality of service (QoS) and the service level agreement (SLA). Algorithms are divided into static and dynamic ones. The static algorithm used by balancer is way of selecting one of these algorithms such as custom load, random and leaky bucket from the beginning and throughout. The dynamic algorithm is to assign a destined node for distribution for any requests based on the collection of node information such as channel bandwidth, node’s utilization or queue length. Correspondingly, it does select the proper node to distribute. This paper focuses only on the investigation of static algorithms as follows.

2.1. Random (RND)

The distribution of tasks is treated randomly. The priority is given randomly provided that equal probability to each node is assigned. The algorithm associates with equal probability of each node. As the destined node avails, it is dispatched to the node immediately.

2.2. Leaky bucket (LB)

The balancer stabilizes any requests in the way that it polices the traffic. The LB algorithm [23] is a mechanism of policing requests and shaping them into a fixed-speed distribution of traffic in the clouds. The LB is to govern the metered-speed of any communication channels then go under the bandwidth limit for a given period of time, in order to cut off an added charge. The algorithm works in such a mechanism of a leaky bucket with liquid. The LB shapes the traffic up to its capacity. The traffic is dispatched from the bucket at a set-speed and fixed-amount. In case of overload, the traffic is considered to be non-conformant.

2.3. Custom load algorithm (CLA)

The algorithm [24] selects the nodes based on its load. This mechanism chooses the less loaded nodes. The load can be computed from the available queue space, utilization, and available bandwidth. The algorithm observes the calculated load of each node. The algorithm is a resourceful method of providing node utilization. It is one of the important algorithms used by the balancer to dispatch the traffic across the cloud systems and guarantee the QoS and reliability.

2.4. Round Robin (RR)

The algorithm [25] serves a request by pondering the fairness by taking the turn to dispatch each task from the queue. Each task is moved out to take a service at the destined node within the fixed time if it is not preemptive then put to the queue’s rearmost so that the first-in-queue task takes on the service instead. RR algorithm is a mechanism used by many applications that oblige multiple users who request for service nodes. It controls any requests in a rounded first-in-first-out (FIFO) manner and ignores the priority setting in order that applications fairly use the services. It is the typical, fairest, modest, and mostly used algorithms, since it is straightforward to apply and regardless of priority level, only a FIFO discipline and a fixed time for each node. The algorithm solves the problem of bandwidth hog where a user’s task occupies resource for a long-lasting time as the preemption is ignored.

3. SIMULATION RESULTS AND ANALYSIS

The queueing network model which represents the cloud system and comprises seven processors is used. Node one is the *splitter* with variable serviced time. Both splitting and merging time are computed randomly, depending on concurrency number. The queue disciplines at all nodes is FIFO. Node two denotes a cloud balancer, while other nodes, server 3 to 6, represent the cloud’s data center with the same branching. These four data centers form parallel session. These application-dependent service time distributions for node 3, 4, 5 and 6 follow exponential distribution with mean of 2, 4, 6, and 8 seconds for A_1, A_2, A_3, and A_4 correspondingly. The simulation model for the experiment is shown in Figure 2. The arrival rate of the application observes Erlang-2 distribution with a mean of 2 in a second. The concurrent applications (A_1, A_2, A_3, and A_4) for the *balancer* improvement exhibit A_1 = [1 2 3 6], A_2 = [4 3 1 0], A_3 = [5 4 5 1] and A_4 = [1 1 6 8]. The simulation [20] is run to investigate load balancing metrics such as mean queue length (MQL), throughput (THR), mean waiting time in queue (MWT) and utilization (UTL).

The simulation results are shown in Tables 1 to 4. The performance metrics of the four algorithms with Erlang concurrent applications on clouds environment: (a) the random, (b) the leaky bucket, (c) the custom load, and (d) the round Robin algorithms are listed in these tables. It can be noted that the leaky bucket seems to outshine highest number of non-conformant data which can degrade multimedia traffic as depicted in Table 2. However, the LB will balance the load well with the lowest load range (= 1.78). Table 3 proves that the CLA improves a highest throughput than other algorithms, particularly in the heterogeneous resources (four datacenters). The RND and RR’s static scheduler algorithms anticipate the heterogeneous Erlang concurrent applications along with processing capability of the data centers. The number of concurrent jobs is assigned to meet the greater capacity on heterogeneous environments in order to help accomplish the jobs in less execution time. However, both RND and RR do not balance the load due to the range of traffic load being about 17.43 and 15.45 as shown in Tables 1 and 4, respectively. The modest RND 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 random fashion. Thus, its achievement time of the concurrent jobs is greater than LB and CLA algorithms.

Table 1. Simulation results from RND algorithm

Random Algorithm (RND)				
Data Center	MQL	THR	MWT	UTL
3	9.58	2.5	12.53	1.0
4	24.8	1.2	5.16	1.0
5	7.37	2.3	8.57	1.0
6	14.96	2	7.26	1.0
Load range	17.43			

Table 2. Simulation results from LB algorithm

Leaky Bucket (LB)					
Data Center	MQL	THR	MWT	UTL	Non-Conformance
3	3.09	2	11.75	0.95	2.2
4	3.63	1.8	11.41	0.99	2.4
5	2.47	1.5	6.89	0.81	2
6	1.85	2.3	7.72	0.91	2.9
Load range	1.78				

Table 3. Simulation results from CLA algorithm

Custom Load Algorithm (CLA)				
Data Center	MQL	THR	MWT	UTL
3	8.39	2.4	9.5	1.0
4	11.83	3	10.46	1.0
5	11.7	2.4	3.96	1.0
6	13.26	2.5	18	1.0
Load range	4.87			

Table 4. Simulation results from RR algorithm

Round Robin (RR)				
Data Center	MQL	THR	MWT	UTL
3	8.95	2.2	12.49	1.0
4	11.86	2.2	13.3	1.0
5	24.4	2.2	10.8	1.0
6	10.5	2.1	10.68	1.0
Load range	15.45			

4. CONCLUSION

In this paper, the load balancing algorithms are investigated to reflect the capacities of each data center and the queue length in each center is studied in order to allocate the Erlang traffic arriving at the balancer into the most proper center. The scheduler algorithms consider the load (queue length) in front of centers. The performance metrics from simulation show that the LB is most appropriate to the Erlang concurrent applications with mixed resources (data centers) compared to the other round robin, random and CLA algorithms. But in regards of the QoS when considering the throughput and the non-conformance as a performance metric as well, the CLA algorithm is the most appropriate. In the future work, the balancer processing time can be taken into account. Besides, the state space calculation of concurrent applications in the system will be investigated for the approximation. In the next step, further estimation to reduce computation time for all algorithms will be taken into account.

REFERENCES

[1] Audzevich Y., Bodrog L., Ofek Y., Telek M., "Packet Loss Minimization in Load-Balancing Switch," *International Conference on Analytical and Stochastic Modeling Techniques and Applications*, Cardiff, pp. 44-58, 2010.

[2] Mirzoev T., Kane S., "What is health systems responsiveness? Review of existing knowledge and proposed conceptual framework," *BMJ Global Health*, vol. 2, no. 4, pp. 1-11, 2017.

[3] Rodriguez A., Trelles O., Ujaldon M., "Using Graphics Processors for High Performance Normalization of Gene-Expression," *IEEE 13th International Conference on High Performance Computing and Communications (HPPC)*, Banff, pp. 599-604, 2011.

[4] Malhotra L., Agarwal D., Jaiswal A., "Virtualization in Cloud Computing," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 8, pp. 745-749, 2014.

[5] Lingayat A., et al. "Integration of Linux Containers in OpenStack: An Introspection," *Indonesian Journal of Electrical Engineering and Computering*, vol. 12, no. 3, pp. 1094-1105, 2018.

[6] Francis T., "A Comparison of Cloud Execution Mechanisms Fog, Edge, and Clone Cloud Computing," *International Journal of Electrical and Computer Engineering*, vol. 8, no. 6, pp. 4646-4653, 2018.

[7] Mahadevan R., Anbazhagen N., "An Efficient Framework to Improve QoS of CSP using Enhanced Minimal Resource Optimization based Scheduling Algorithm," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 12, no. 3, pp. 1179-1186, 2018.

[8] Tarahomi M., Izadi M., "A hybrid algorithm to reduce energy consumption management in cloud data centers," *International Journal of Electrical and Computer Engineering*, vol. 9, no. 1, pp. 554-561, 2019.

[9] Zaidi T., Rampratap R., "Virtual Machine Allocation Policy in Cloud Computing Environment using CloudSim," *International Journal of Electrical and Computer Engineering*, vol. 8, no. 1, pp. 344-354, 2018.

[10] Ngenzi A., Selvarani R., Suchithra R., "FDMC: Framework for Decision Making in Cloud for Efficient Resource Management," *International Journal of Electrical and Computer Engineering*, vol. 7, no. 1, pp. 496-504, 2017.

[11] Mohazabiyeh A. R., Amirizadeh K. H., "Energy-Aware Adaptive Four Thresholds Technique for Optimal Virtual Machine Placement," *International Journal of Electrical and Computer Engineering*, vol. 8, no. 5, pp. 3890-3901, 2018.

[12] Bhawiyuga A., et al. "Architectural design of IoT-cloud computing integration platform," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 3, pp. 1399-1408, 2019.

[13] Ibrahim A. K. M. et al. "Lightweight IoT middleware for rapid application development," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 3, pp. 1385-1392, 2019.

[14] Xing Y., Zhan Y., "Virtualization and Cloud Computing," *International Conference on Future Wireless Networks and Information Systems*, Macao, pp. 305-312, 2012.

[15] Velde V., Rama B., "A Virtual Machine Based Load Balancing Algorithm for Cloud Computing," *International Journal of Pure and Applied Mathematics*, vol. 117, no. 21, pp. 219-229, 2017.

[16] Venkatachalapathy K., Thiyagarajan V. S., Ayyasamy A., Ranjani K., "Big Data with Cloud Virtualization for Effective Resource Handling," *International Journal of Control Theory and Applications*, vol. 9, no. 2, pp. 435-444, 2016.

[17] Conte A., Scarsini M., Sürücü O., "An experimental investigation into queueing behaviour," *Jena Economic Research Papers*, vol. 2014, no. 30, pp. 1-20, 2014.

[18] Chromy E., Suran J., Kovacik M., Kavacky M., "Usage of Erlang Formula in IP Networks," *Communications and Network Online Journal*, vol. 3, no. 3, pp. 161-167, 2011.

[19] Ashtari A. et. al. "Using a Priori Information for Regularization in Breast Microwave Image Reconstruction," *IEEE Transactions on Biomedical Engineering*, vol. 57, no. 9, pp. 2197-2208, 2010.

[20] Behrokh Khoshnevis, Contour Crafting Corporation. [Online]. Available: http://www.bkhoshnevis.com/.

[21] Li J., Lin X., Nazarian S., Pedram M., "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] Swarna M., Ravi S., Anand M., "Leaky Bucket Algorithm for Congestion Control," *International Journal of Applied Engineering Research*, vol. 11, no. 5, pp. 3155-3159, 2016.

[23] Paez G., Amaya J., "Traffic management algorithms: Leaky Bucket, Token Bucket and Virtual Scheduling," *Tecnura*, vol. 15, no. 29, pp. 76-89, 2011.

[24] Devi D. C., Uthariaraj V. R., "Load Balancing in Cloud Computing Environment Using Improved Weighted Round Robin Algorithm for Nonpreemptive Dependent Tasks," *The Scientific World Journal*, vol. 2016, pp. 1-14, 2016.

[25] Panda S. K., Bhoi S. K., "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.