SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 15, No. 1, February 2018, 85-95

UDC: 004.779:004.72 **DOI: https://doi.org/10.2298/SJEE1801085B**

A Simulation of Distributed STM

Dragan Brkin[1], Branislav Kordić[2], Miroslav Popović[1]

Abstract: This paper presents an extension of the IaaS Cloud simulator CloudSim. Computational tasks are modeled in the form of a transaction on a transactional memory and communication between the data center is based on the Two-Phase Commit protocol. The model of the distributed STM prototype is implemented using the extended CloudSim simulator. The obtained results are as expected and in accordance with desired system behavior. The presented results are positive and they stimulate future work in development of distributed STM.

Keywords: CloudSim, Software Transactional Memory, Distributed Systems, Two phase commit protocol.

1 Introduction

Transactional Memory (TM) is one of the mechanisms of parallel programming for multi-core architecture. Software Transactional Memory (STM) is software implementation of TM paradigm, while in the similar manner Distributed Transactional Memory (DTM) generalizes the same mechanism for Distributed systems [1, 2].

Cloud Computing is a technology that offers a new way of hardware infrastructure utilization. Cloud providers offer these infrastructures as a virtual hardware managed by the appropriate software. They offer their services in the form of Virtual Machines (VM) ready for use on users demand [5] making a good base for distributed systems.

Infrastructure as a Service (IaaS) is one of the three basic models in the cloud computing. This model allows the use of computer infrastructure in the form of VM. A user is able to manage one ore more VMs, their networking as well as data storage. In a virtual infrastructure, the user can run various types of program support, from operating systems to applications. The Internet is used to access the infrastructure. In order to make this service available to users, software infrastructure has to provide the infrastructure administration, the resource allocation, the infrastructure management, and the performance measurement.

[1]Faculty of Technical Sciences in Novi Sad, University of Novi Sad, Trg Dositeja Obradovića 6, 21000, Novi Sad, Serbia; E-mails: dragan.brkin@live.com; miroslav.popovic@rt-rk.uns.ac.rs
[2]RT-RK Institute for Computer Based Systems, Narodnog fronta 23a, 21000 Novi Sad, Srbija; E-mail: branislav.kordic@rt-rk.com

CloudSim is one of the best known cloud computing and simulation tools for modeling infrastructure as a service [3]. The CloudSim tool is used for simulation and analysis of large-scale applications, such as cloud social networks [7], evaluation of SLA-based VM scheduling strategies for cloud federations [8], and for performance and energy estimation with integrating I/Os in Cloudsim [6]. According to the best of our knowledge, this is the first work that contributes to the modeling and simulation of distributed STM-based systems.

The two-phase commit protocol (2PC) is a continuous update protocol used in distributed systems. The protocol, as the name itself indicates, consists of two phases. The first phase represents an update request that the transaction handler (coordinator) sends to all the transaction resources in order to determine whether the transaction will be executed or terminated. The second phase is the execution or termination of a transaction, depending on whether the response, which transaction coordinator gets at the request, is positive or negative, respectively.

This paper consists of six sections. The second section describes CloudSim, a tool used to simulate a distributed system. In the third section the model and the simulated system architecture are described. The fourth section illustrates the example of the system behavior. The implementation details are presented in the fifth section. In the last section the evaluation of the implemented model and results are shown.

2 CloudSim Tool

CloudSim is a tool for modeling and simulating a cloud computing environment and evaluating resource management algorithms [6 –7]. This tool works as a simulator based on discrete events and is implemented in Java programming language. The basic elements of CloudSim are the following entities: *CloudInformationService*, *Datacenter*, *DatacenterBroker*, and *CloudsimShutdown*. These entities represent the basics of Datacenter architecture. The communication between entities is accomplished by sending defined event messages (e.g. VM_CREATE, CLOUDLET_SUBMIT etc). These events can be external (sent by one entity to another) or internal (sent and received by the same entity). Upon receipt, each of the event messages will be received and certain actions will be performed before the confirmation message is sent (e.g. VM_CREATE_ACK) [6]. The CloudSim simulation is based on running the *Cloudlet* class object. The *Cloudlet* model describes a process determined by the size of the input and the output data, the required RAM, the number of processor cores, the processor loads, the scheduling mode, etc. The cloudlet is executed on a VM within the data center.

Broker is the model of the end-user, i.e. client that starts transactions, i.e. it sends requests for transaction to be processed on the VM and accepts the processing results.

3 CloudSim Model of PSTM System

This section describes the designed architecture of the simulated system. Then the description of the corresponding CloudSim model is given.

3.1 Architecture of PSTM system

Python Software Transactional Memory - PSTM was implemented using a dictionary within the data center. A copy of the transactional variable (t-variable) is forwarded to each user upon request. And, if possible, the dictionary state is updated on the user demand. A queue is used so that the requests can be processed in the order they have been sent [5]. A client using transactional memory is called a TM application.

A PSTM-based application contains a set of transactions that perform operations over local variables, i.e. copies of t-variables. The transaction requires copies of t-variables at the beginning and updates some of them at the end, using the PSTM API interface. This interface is a set of public functions defined in the PSTM module where the request for processing is sent to a remote server using the Remote Procedure Call mechanism (RPC). The dictionary used is a collection of tuples (*key*, *ver*, *val*), where *key* is the name of the t-variable, *ver* is the current version, and the *val* is the value. The PSTM server uses incoming requests automatically. Upon receipt, the request is processed and the response is immediately sent back to the client over a pipe (each request has its own pipe) [4].

3.2 CloudSim model

The architecture of the simulated distributed system consists of several servers presented in [4]. The 2PC protocol enables the coordinator to communicate with data centers.

At the very beginning of model design, the assumption was made that the communication between the data centers is negligible in terms of time in relation to the processing of the transaction. In order to simplify the model and simulation of transaction processing, it was considered that all data centers in the system have the same data. It should also be noted that each transaction is performed on a separate VM either at the same or different data center. After testing and determining the correctness of the system, the initial assumption of the processing time and time needed for transaction to be sent and received as well as for data center communications was eliminated. More realistic parameters have been put forward. As the communication in reality is much slower than the processing itself, the correctness of the system was again put to

the test. After testing and simulating different scenarios, the obtained results are positive, which indicates a properly designed system. The CloudSim model block diagram is shown in Fig. 1

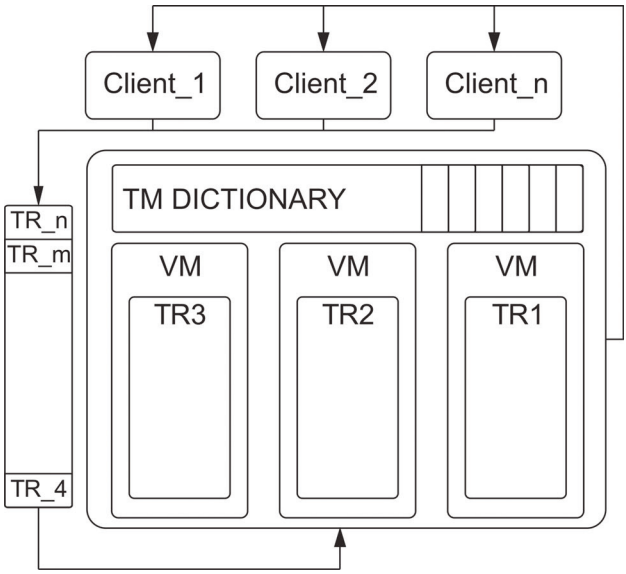

Fig. 1 – *Block diagram of CloudSim model.*

Transaction model: represents the extension of the Cloudlet class, which, as already mentioned, models the process that is executed on the VM. The transaction is defined by the type of transaction, the current status, and the sets of t-variables on which the *read/write* operations are performed. The transaction is a continuous operation and does not depend on the type of transaction, nor on the size of the t-variables sets on which *read* or *write* operations are performed.

The STM server dictionary model: A dictionary used in a distributed system is modeled as a pair (key, ver) where *key* represents the unique t-variable key, and ver the current version. The dictionary from [4] is simplified because the value of the t-variable for system simulation is not relevant. The simulation also does not include possible network congestion as well as failures of the servers on which the transaction is processed.

The 2PC protocol model: In order for the 2PC protocol to be implemented, each of the data centers in the simulation must have the address of the coordinator, and the coordinator must have the addresses of all data centers. The address list must be delivered to the coordinator, as well as its address to all data centers, before the simulation starts, so that at the start time, within the

simulation, all the data centers, and the coordinator have all the information needed to execute the 2PC.

4 Model Behavior

This section provides an example of transaction execution in a distributed environment.

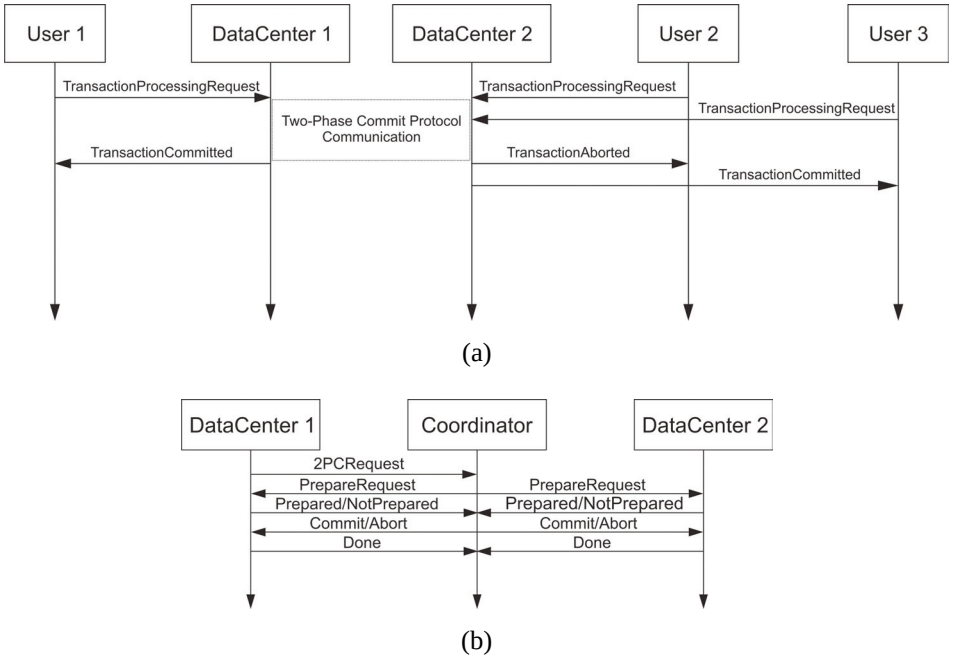

Fig. 2 – *(a) MSC diagram of model behavior. (b) MSC diagram of two-phase commit protocol.*

Fig. 2a shows two data centers to which three users (*User_1*, *User_2* and *User_3*) send requests for transaction processing at about the same time. *User_1* and *User_2* in this case try to update the same set of t-variables, while *User_3* performs reading and validating the different set of read values. On one server, i.e. data center is processing request sent by *User_1*, while requests sent by *User_2* and *User_3* are processed by another. As the *User_3* only validates previously read independent set of t-variables it is not in conflict with any other transaction, so it is successfully executed. However, *User_1* and *User_2* attempt to update the same set of t-variables and they come into conflict at 2PC. As both of these requests can be processed locally, both sides send 2PC request to the Coordinator. Request from *Data_Center_1* has reached first to the coordinator, and it will be processed before the request from *Data_Center_2*.

Coordinator first sends PrepareRequest message to all and waits for answers (*Prepared/NotPrepared*). After that, it sends a command (*Commit/Abort*), according to the previously received answers, Fig. 2b.

In this case, the update, i.e. commit is performed by *Data_Center_1* and then alignment of the local dictionary in the servers is performed. After an unsuccessful update, the *User_2* requests a set of t-variables in order to have the latest versions of t-variables that he wants to update.

It is important to note that data centers communicate with the coordinator when an update, i.e. commit of t-variable is required. All communication between data centers takes place through the coordinator.

5 Implementation

In the implementation process, the already existing CloudSim class has been extended with minor changes to the original implementation. Fig. 3 shows the UML diagram of extended classes. The diagram shows only extended and subsequently added classes necessary for simulation and transaction handling.

The *Tvar* class represents the object of t-variable, which is the basic building element of a transaction. In order for a transaction to be sent to the data center by a broker, or VM, the corresponding event messages and operators were introduced for these events, thus enabling the processing of the transaction to the VM and returning the processing results to the broker. All event handlers required to send and receive processed transactions are implemented in the *TransactionDatacenterBroker* class.

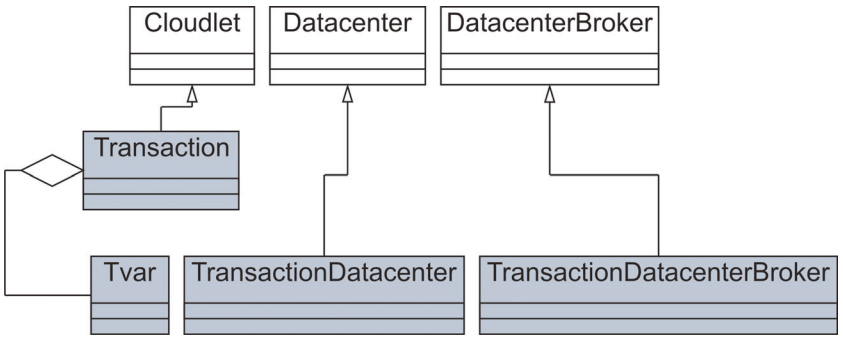

Fig. 3 – *UML diagram of CloudSim extended classes.*

The Transaction class contains the implementation of the aforementioned transaction model. The object of this class is an integral part of the message sent by the broker to VM and vice versa. A transaction can perform read and/or write operations.

The data center has local hosts containing VMs where transactions are executed. In order for transactions to be executed on VM, it was necessary to extend the original implementation of the *Datacenter* class by adding new transaction processing event messages, as well as the messages necessary for the implementation of the 2PC protocol. The operators for these events are implemented in the *TransactionDatacenter* class.

6 Evaluation and Results

This section provides an evaluation of the system and the results of the simulated relevant cases. All results are shown in corresponding tables. The results provides information regarding transactions start time (*Start*), finish time (*End*), read and write operations (*Read/Write*), and a number of transaction aborts (*Aborts*). Times in tables are given in an unnamed time unit provided by Cloudsim.

Table 1
Results of a group of transactions performing read or write operations at the same data center.

ID	Start	End	Read/Write	Aborts
TR1	0.1	1600.1	5/0	0
TR2	0.1	1600.1	5/0	0
TR3	0.1	1600.1	0/5	0
TR4	1600.1	3200.1	0/5	1
TR5	3200.1	4800.1	0/5	2

The duration of the transaction in the simulation is calculated based on the defined characteristics of the process. Each transaction is defined, as already mentioned, by the processor unit on which it is executed and the number of instructions necessary to execute. The number of instructions is expressed in terms of MI (Million Instruction). VM is also defined by the number of processor units, where each processor unit is defined by the MI [3]. Based on the characteristics of the VM and the defined process, CloudSim automatically calculates the time required for transaction to be processed.

Table 1 shows the results of a group of transactions carried out at a single data center and all transactions are initiated at the same time. Transactions TR1 and TR2 perform a read operation on one set of t-variables, while transactions TR3, TR4 and TR5 perform the write operation on another. A intersection of a set of t-variables that TR1 and TR2 read, and t-variables that TR3, TR4, and TR5 are trying to update is an empty set. As transactions TR3, TR4 and TR5 work over the same set of t-variables, there is a conflict between them. Among the transactions that are in conflict only one can be successfully done, while others are unsuccessful. In this case, transaction TR3 was executed successfully

and it is executed in parallel with the transactions TR1 and TR2 that perform the read and validate operation on different set of t-variables. The *Start* and *End* time of the TR4 and TR5 transactions is the consequence of previous unsuccessful commits. Each unsuccessfully completed transaction is followed by the request for the latest version of the t-variables set it wants to update.

Table 2
Results of a group of transactions performing read and write operations at various data centers.

ID	Start	End	Read/Write	Aborts
TR1	0.1	1600.1	5/5	0
TR2	1600.2	3200.2	5/5	1
TR3	3200.3	4800.3	5/5	2
TR4	4800.3	6400.4	5/5	3
TR5	6400.4	8000.5	5/5	4

Table 2 presents the results for a group of five transactions that perform both *read* and *write* operations, each at a different data center. All transactions use the same set of t-variables and start at about the same time. Some small deviations may be noticed in the execution time. These deviations are the result of Cloudsim introduction of discrete time delays between events. Each of the transactions attempts to update the previously read set of t-variable, so therefore all transactions are in conflict. This conflict affects the time needed for the transaction to be completed. The second transaction took twice longer than the first one, which is the result of the first unsuccessful update.

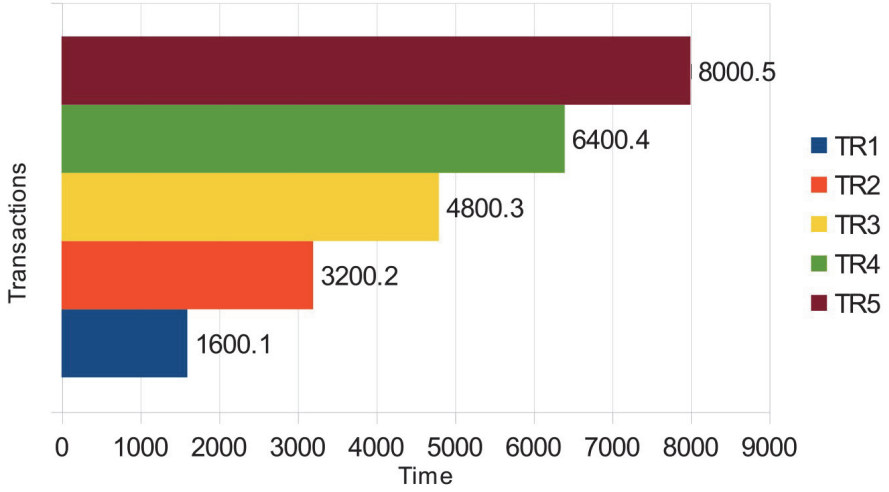

Fig. 4 – *Graphical representation of transaction execution in time.*

Fig. 4 presents the graphical representation of transaction execution TR1-TR5 from **Table 2**. As it can be seen from the graphics, the duration of the transaction is directly proportional to the number of previously unsuccessfully executed transactions.

As it was mentioned before, after verification of the correctness of the algorithm for processing transactions and communication between the distributed centers, parameters that define a transaction have been corrected. Dependence between the number of t-variable, as well as their type, over which transaction is performed and the duration of transaction processing on VM has been brought into simulation. The dependence between the number of t-variable and the time required for the transaction to be sent to the VM has been brought in as well.

Fig. 5 – *Simulated clique topology.*

Graphical representation of simulated clique topology is shown on Fig. 5. Details of this specific topology, such as length of link from one node to another and its delay is presented in **Table 3**.

Table 3

Links details of simulated clique topology.

From	To	Euclidean length	Propagation delay
North America	South America	10655	28.156
North America	Europe	9128	32.866
South America	Asia	17674	16.974
South America	Australia	13202	22.724
Europe	South America	9579	31.322
Europe	Asia	9142	32.816
Asia	North America	11086	27.061
Asia	Australia	7427	40.393
Australia	North America	12649	23.717
Australia	Europe	16311	18.392

All links have the same bandwidth. Euclidean length and propagation delay have been given in the unnamed unit. Each node is located on a separate continent.

The time required for transaction to be processed does not depend only on the number of t-variables and desired operation, but also on the network. The network is much slower then data center, and because of that the time needed for transaction to be processed is most dependent on the network delay, as well as of the distance from user to the data center.

Table 4
Results of a group of transactions performing read and write operations at various data centers.

ID	Start	End	Read/Write	Aborts
TR1	22.54	66.54	50/50	0
TR2	90.37	134.37	50/50	1
TR3	158.92	202.92	50/50	2
TR4	227.82	271.82	50/50	3
TR5	297.72	341.72	50/50	4

Table 4 presents the results for a group of five transactions that perform both *read* and *write* operation, each at a different data center. The data centers and the users are connected in the clique topology as it is shown in Fig. 5. The congestion and server failures were not considered in this example. The difference in the transaction processing times is the result of a network delay that differs for each node in the network. Similar to the previous example all the transactions use the same set of t-variables and start at about the same time. Each of the transactions attempts to update the previously read set of 50 t-variables, so therefore all transactions are in conflict. Column *Start* in the table shows the time when certain transaction has started and committed successfully.

Because the coordinator is located inside one of the data center, request for 2PC protocol from that data center will be first processed. As it can be expected, request from the most distant data center will be last processed, but that is the case only if every data center sends its request at the same time. As it was mentioned before, after unsuccessful update, the data center, i.e. user must get the latest t-variables which it wants to update. From this moment it cannot be determined precisely when the data center, i.e. user, will send its next request.

7 Conclusion

This paper presents CloudSim simulation of the distributed STM system. The way the simulator operates and the behavior of the system is presented. Then simulation model of a prototype of distributed STM is implemented. Evaluation was done through the simulation of the most critical cases that

would impair the proper functioning of the system. The obtained results verified the correctness of the algorithm for processing transactions and communication between the distributed centers. Further work could show the effect of stochastic network congestion on system performance.

8 Acknowledgment

This work is partly financed by the Ministry of Education, Science and Technological Development of the Republic of Serbia, on project No. III_044009_2.

9 References

[1] D. Brkin, B. Kordic, M. Popovic: A Solution of CloudSim Simulation of Distributed STM, 61st Meeteng of the Society Society for Electronics, Telecommunications, Computers, Automatic Control and Nuclear Engineering (ETRAN), 05-08 June 2017, Kladovo, pp.RT3.4 1 − 4

[2] T. Harris, J. Larus, R. Rajwar: Transactional Memory, 2nd edition, Morgan and Claypool, Williston, 2010.

[3] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya: Cloudsim: A Toolkit for Modeling and Simulation of Cloud Computing Enviroments and Evaluation of Resource Provisioning Algorithms, Software: Practice and Experience, Vol. 41, No. 1, 2011, pp. 23 − 50.

[4] M. Popovic, B. Kordic: PSTM: Python Software Transactional Memory, 22nd Telecommunications Forum Telfor (TELFOR), November 2014, Belgrade, pp. 1106 − 1109.

[5] B. Kordic, M. Popovic, I. Bašičević: DPM-PSTM: Dual-port Memory Based Python Software Transactional Memory, 4th Eastern European Regional Conference on the Engineering of Computer Based Systems, August 2015, Belgrade, pp. 126 − 129.

[6] H. Quarnoughi, J. Boukhobza, F. Singhoff, S. Rubini: Integrating I/Os in Cloudsim for Performance and Energy Estimation, ACM SIGOPS Operating Systems Review - Special Topics, Vol. 50, No. 2, 2016, pp. 27 − 36.

[7] B. Wickremasinghe, R. N. Calheiros, R. Buyya: CloudAnalyst: A CloudSim-based Visual Modeller for Analysing Cloud Computing Environments and Applications, 24th IEEE Internationa Conference on Advanced Information Networking and Applications (AINA), April 2010, pp. 446 − 452.

[8] A. Kohne, D. Pasternak, L. Nagel, O. Spinczyk: Evaluation of SLA-based Decision Strategies for VM Scheduling in Cloud Data Centers, Proceedings of the 3rd Workshop on CrossCloud Infrastructures & Platforms, CrossCloud ’16, April 2016, No. 6, pp. 6:1 − 6:5.