GetLB++: Improving Transaction Load Balancing on the Electronic Funds Transfer Landscape

Felipe Rabuske[1], Cristiano André da Costa[1], Rodrigo da Rosa Righi[1], Gustavo Rostirolla[1], Antonio Alberti[2], Anselm Busse[3] and Hans-Ulrich Heiss[3]

[1]Applied Computing Graduate Program - Univ. Vale do Rio dos Sinos - São Leopoldo, RS, Brazil

[2]Instituto Nacional de Telecomunicações - INATEL - Santa Rita do Sapucaí, MG, Brazil

[3]Technical University Berlin - TU Berlin - Sekretariat EN 6 - Einsteinufer 17, D-10587 Berlin

Abstract— *The continuous growth of electronic payment methods in business transactions is a reality that boosts e-commerce, being present more and more in our daily lives. Considering this situation, we proposed in a previous work a model denoted GetLB, which contains a scheduler that provides good results when compared to the traditional dispatching approach – the Round-Robin. Although offering a good distribution of transactions to processing (PMs), GetLB's scheduling routine is time consuming since each input is always analyzed against each target PM. Thus, in this paper we are proposing GetLB++ – a GetLB improvement that covers scheduling computation efficiency by bursting a transaction to a specific PM in accordance with a updated in-memory decreasing-sorted list of PM capacities. The results using an Amazon EC2 cluster instance showed a higher scheduling speed on GetLB++ in comparison with the standard GetLB, presenting gains about 20% on the makespan time. Besides scalability on EFT systems, GetLB++'s contributions are not limited to the context of transactional systems, but can also be extended for load balancing in e-commerce systems, cloud computing, and parallel programming.*

Keywords: Eletronic Funds Transfer, Scheduling, Algorithm

1. Introduction

Electronic payment methods, such as debit and credit cards, are being adopted by the society as the mainstream payment method for business transactions [1]. The benefits offered by EFT (Electronic Funds Transfer) range from a higher commodity for the buyers to a greater security for commercial institutions [2]. Usually, the dispatcher that receives transactions in the EFT company uses the Round-Robin (RR) algorithm to distribute them to processing machines, or PMs [3], [4]. RR algorithm consists of dispatching the new tasks in a circular fashion amongst the PMs, guaranteeing that the tasks are distributed uniformly between the processing units [5]. RR can be seen as a very fast strategy, with complexity $O(1)$, presenting an optimal load balancing for homogeneous systems [6]: when both consumers (in our case, electronic transactions) and resources (in our case, PMs) have the same configuration, performance is kept unchanged over time. Nevertheless, this scenario is not common in EFT systems, because of each kind of transaction has different computational needs [2].

Regarding the aforementioned scope, we developed a model named GetLB, which proposes a framework for scheduling transactions on an EFT company [7]. Periodically, GetLB's scheduling algorithm takes into account several characteristics of the transactions, such as the number of CPU instructions and memory consumption, as well as PMs data, to distribute the tasks. According to [7] GetLB obtained good results when distributing the workload amongst a dozen of nodes on homogeneous and heterogeneous clusters. The processing time of the GetLB's scheduling algorithm was about seven times greater than the RR routine. Additionally, this time tends to be bigger as the number of PM nodes of a cluster increases as well, facing a scalability problem. This happens for two reasons: (i) PMs periodically update their data to the dispatcher; (ii) at each transaction input, GetLB recalculates the workload that would be added on each PM of the cluster before choosing the one that will receive a new transaction.

Clearly, there is a gap on GetLB that can be explored in terms of expanding the model's scalability. In this way, we have developed GetLB++ — an enhanced and more flexible version of the standard GetLB, now focusing on improving scheduling routine (*i.e.*, the calculus involved in the scheduling procedure) but maintaining or yet improving the quality of the transactions-PMs assignment. GetLB++ can be used for processing different types of tasks, not being restricted to EFT scenarios. We developed a prototype that covers both GetLB and GetLB++ algorithms, besides an implementation of the RR. The prototype was evaluated with different rates for transactions arrival, distinct Amazon EC2 cluster instances and various input workloads. The results were encouraging, where GetLB++ obtained a better scheduling time and quality, besides presenting a larger scalability when compared to GetLB and RR.

The remainder of this article will first introduce the fundamental concepts in Section 2, presenting the main ideas of GetLB. Section 3 discusses about the related studies, giving the open issues in the EFT area. Section 4 describes the GetLB++ model in details, while Section 5 covers its

prototype. Section 6 brings the evaluation methodology to analyze GetLB, GetLB++, and RR. Section 7, in turn, discusses the results obtained from the experiments. Finally, the conclusion is written in Section 8, addressing scientific contributions and future work.

2. Background

This section presents the functioning of GetLB [7], offering the basis to understand the advances in the newer version. Electronic funds transfer transactions have different processing needs: CPU time, database access, network, and access to external systems. Thinking about these peculiarities, GetLB was developed to deliver a better load balance for transactions to PMs, so enabling benefits both to the users and provider company administrators. Figure 1 depicts the GetLB's architecture. GetLB was structured with the following design decisions in mind: (i) the scheduling heuristic algorithm runs in the dispatcher module and must work with up to date information regarding the PMs; (ii) the heuristic scheduling must combine relevant data in order to compose the notion of load; (iii) PMs must be capable to notify the switch; (iv) the framework must deal with heterogeneous resources at both communication and computing levels.

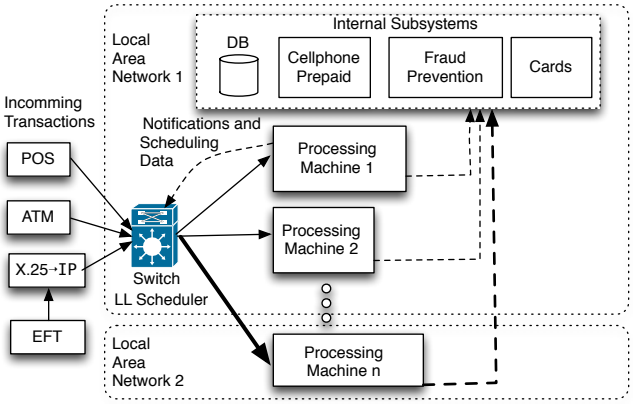

Fig. 1: GetLB architecture, emphasizing network decoupling and "PMs→switch" cooperative interaction besides the traditional one for transaction dispatching in opposite direction.

Regarding the scheduling activities, the dispatcher has an array that contains information about all PMs. Processing machines periodically report updates to the dispatcher, impacting on updating its array afterwards. The dispatcher performs all scheduling calculus with in-memory data, where the updating period informs how recent is PMs data regarding CPU, memory, and network. Thus, we previously developed a scheduling heuristic called LL (Load Level), which considers transactions and PMs are heterogeneous and, PMs as a part of a dynamic environment. LL can be viewed as a decision function $LL(i,j)$ where i means a specific type of transaction, while j denotes a candidate target PM for receiving transaction i. For each new transaction i, the switch will calculate n equations $LL(i,j)$, where n means the number of processing machines. Therefore, the lowest result will inform the target that will receive a transaction. $LL(i,j)$ can be obtained by computing Equation 1.

$$LL(i,j) = Recv(i,j) + Proc(i,j)\,, \tag{1}$$

$$Recv(i,j) = bytes(i) \times transfer(j)\,, \tag{2}$$

$$Proc(i,j) = transaction(i,j) + \sum_{z=0}^{m-1} transaction(z,j)\,, \tag{3}$$

$$\begin{aligned} transaction(i,j) &= \frac{instructions(i)}{clock(j) \times [1 - load(j)]} \\ &+ \frac{RAM(i) \times serviceRAM(j)}{freeRAM(j)} \\ &+ \frac{HD(i) \times serviceHD(j)}{freeHD(j)} \\ &+ sub(i,j)\,, \end{aligned} \tag{4}$$

$$sub(i,j) = \sum_{y=0}^{x-1} [2\, sub_a(y,j) + sub_c(y)] \times sub_r(i,y)\,. \tag{5}$$

Equation 1 is given by adding the estimated reception time ($Recv(i,j)$) and the estimated processing time ($Proc(i,j)$) of transaction i by machine j. In Equation 2, $bytes(i)$ means the size of the transaction i and $transfer(j)$ refers to the time necessary to transfer a single byte to the PM j. The Equation 3 calculates the total time that machine j needs to process transaction i, being divided in two sub-elements: (i) a prediction of computation time for transaction i on PM j; (ii) a prediction of all m transactions that have already been mapped to PM j previously and remain on its input queue. Static data (theoretical values for CPU, memory and access time to sub-systems) and the machine's dynamic data (considering CPU load, communication time and I/O requirements) are taken into consideration to calculate the estimated processing time, represented here in Equation 4.

Equation 5 captures the time spent by the sub-systems accessed by the machine j in order to process the transaction i. Each type of transaction i must access x subsystems. Thus, $sub_a(y,j)$ considers the time spent by PM j for accessing the particular subsystem y through network interaction. This time is multiplied for 2 in order to consider a round-trip evaluation. The field $sub_c(y)$ refers to the service time of the subsystem y and $sub_r(i,y)$ represent the number of times that subsystem y is called for the complete computation of i. The main drawback of the GetLB's algorithm is explained as follows: considering empty PMs, a single task i is mapped to PM0 so we have a $LL(i,0)$ equal to x. For the next transactions that test j as destination, the previous mapped transaction i can impact much larger than x since the $load(j)$ in Equation 4 was updated. This feature has a strong impact on limited machines, since they will be set as overloaded faster.

3. Related Work

The most studied topic in electronic transactions systems considers the security of information [8], [9], [10], [11].

However, security is not the only important topic in the context of EFT systems, but we can contemplate the load balancing (and also scheduling and resource management) problem too. This addresses how fast a provider computes a set of transactions, impacting directly in the user experience. In this way, Sousa et al. [12], [3] present a stochastic model for performance evaluation and resource planning. The tests compared measures of disk and processor utilization on a real system against the values obtained through the utilization of the proposed evaluation model. Desnoyers et al. [13] developed a system called Modellus, which allows modeling the usage of data centers around the Internet, automatically. The service rates are typically variable, limiting queuing theory application to this problem.

Mcheick et al. [14] explain that distributed systems can suffer from degradation problems in terms of performance and scalability. The authors point out that static algorithms work fine when there are no variations in the workload; therefore, they are not indicated for EFT scenarios where workload is not know in advance. Righi et al. [7] proposed the GetLB model aiming at filling the aforementioned gap, presenting both a framework and a scheduler capable to handle heterogeneous workload in dynamic environments. Although the heuristic used by GetLB (named Load Level - LL) took around six times more processing time than Round-Robin. In this perspective, we envision an opportunity to address EFT scalability by improving our previous work with the GetLB++ proposal, which is described in the next section.

4. GetLB++: An Improved Model for EFT Transactions Processing

This section presents GetLB++, which provides an evolution of GetLB to improve quantity and speed of the EFT transactions scheduler. GetLB++ consists of two parts: (i) a task processing framework for distributed systems; (ii) a transaction scheduling algorithm. Our idea is to offer a scalable system when combining both aforementioned parts. In terms of architectural elements, as GetLB does, GetLB++ works with input transactions from EFT terminals, a dispatcher or switch that schedules them to end processing units, named as processing machines or PMs. GetLB++ was developed with the following design decisions in mind: (a) the communication from the dispatcher to PMs must be asynchronous to void network latency; (b) PMs must be able to notify the dispatcher of any event that has impact on scheduling decisions; (c) the framework must allow the usage of different load balancing algorithms for scheduling purposes; (d) the framework must be able to process other types of tasks, so not restricting it to electronic funds transfer scenarios.

The dispatcher uses only in-memory data to schedule a transaction to a PM. Therefore, the processing machines are in charge of both monitoring their hardware and updating this information to the dispatcher in accordance to two modes: periodical and critical. In the periodical mode, PMs have a parameter named *verification period* that is used to both check their own hardware status and to send this information to the dispatcher afterwards. In this way, the dispatcher operates locally with data that are update in accordance with the aforesaid parameter. In the critical mode, a verification period is also used, but here only to check the hardware status in the PM. The switch is only updated with the hardware information of a certain processing machine when there is a critical change in the PM context. Thus, considering two consecutive measurements, there is another parameter named *critical change* indicating the percentage to consider as critical a sudden modification in the hardware status, which may represent impact on scheduling procedures.

4.1 Framework Modeling

GetLB++ follows the same idea of architectural elements from the GetLB model, with transactions, EFT terminals, a dispatcher and PMs (see details in Figure 1). In this work we are generalizing the use of transactions by employing the term Task, since GetLB++ was modeled for being not restrictive to the EFT scenarios. The task interface specifies methods that return information about the workload, estimated size, and a list of external systems that are accessed during the task processing. In addition, an object of this type must also implements a method *process()* which actually performs the processing of the task. Through this encapsulation, the terminals can send different types of tasks to be processed by the GetLB++, since the components of the framework do not need to know the implementation of the task. Therefore, this allows the framework to be extended beyond the EFT scenarios.

The GetLB++ model defines that the switch should have the capacity to operate using different scheduling algorithms, where an implementation of the Scheduler interface accomplishes this objective. This interface specifies the *getNextPU()* method, which returns the machine that will receive a task. The scheduling algorithms that implement this interface have access to information related to the processing machines through the list of ProcessingUnits inside the dispatcher, including their task queues. Therefore, when dispatching a task, the switch calls the *getNextPU* method, passing as parameter the list of Processing Units and the task that will be processed, and the scheduler returns the most suitable Processing Unit to accommodate the task.

4.2 Scheduling Algorithm

Although GetLB++ accepts various scheduling algorithms, the framework presents a default scheduler that was completely redesigned in order to meet the following goals when compared to the original GetLB: (i) have a higher

scheduling velocity; (ii) be more scalable; (iii) maintain or improve the load balancing quality by using the same system metrics as GetLB. As depicted in Figure 2, GetLB always computes n times the $LL(i, j)$ function, where n refers to the number of PMs, so verifying the impact and the conclusion time of task i on PM j. Considering that the mean number of already mapped transactions on each PM is equal to m, GetLB schedules a transaction with $O(n.m)$ complexity. See Equations 1 up to 6 for details. On the other hand, as mentioned in Section 1, Round-Robin is not suitable for heterogeneous systems but offers a $O(1)$ complexity.

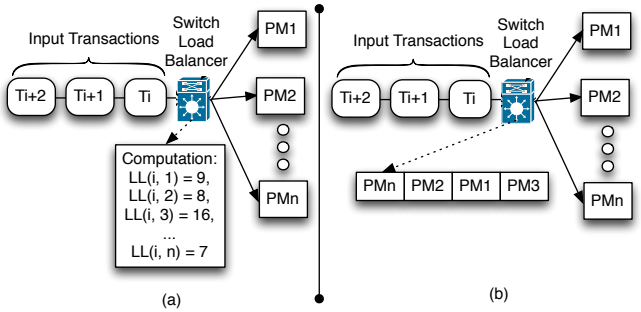

Fig. 2: (a) GetLB scheduling approach, in which each transaction is computed against each PM (Processing Machine); (b) GetLB++ list-scheduling approach.

The GetLB++'s scheduling algorithm goal is to explore scalability on scheduling calculus. Instead of on-the-fly computing the LL indexes and taking again the values for each PM at each incoming task, GetLB++ maintains a descending-ordered resource list which informs the PM with the higher processing capacity at a given moment (See Figure 2 (b)). This allows the system to determine the machine that should process the new task with almost no additional calculations, since the PM on the top of the list will always receive it. The aforesaid list is created when initializing the environment by using the *InitializingPUList* method of the scheduler.

The workload of a task is calculated by Equation 6, where i means the input task and j refers to the top machine in the aforementioned list. The terms *Recv(i,j)* and *transaction(i,j)* were further explained in Section II. The main difference from Equations 1 and 6 is that the last does not take into consideration the already mapped transactions to PM j but only the impact of task i on it. This modeling happens because GetLB++ does not try to test several PMs, but only the impact of a single transaction in a particular PM j. After calculating the workload of this task using *LL'(i,j)*, this value is added to the total load of machine j, that is repositioned in the Processing Unit list afterwards.

$$LL'(i, j) = Recv(i, j) + transaction(i, j) \quad (6)$$

To ensure that the total workload value on a machine reflects its most current state, the workload added by a new task using Equation 6 is stored in the task structure. When this task is completed, the processing machine sends an ending confirmation to the dispatcher. This last subtracts the load value related to the task from the Preprocessing using the *RemoveTask()* method and relocate again the PU in the list of machines.

Finally, we highlight that the GetLB++ scheduler denoted LL' calculates the Load Level of a task against a single PM, no matter the number of PMs and tasks residing in the cluster, which makes the algorithm highly scalable. Unlike this, the standard GetLB algorithm recalculates the Load Level for all machines in the cluster, always considering the target task and all previous mapped tasks on each machine.

5. GetLB++ Prototype

We developed a GetLB++ prototype using Java programming language and RMI middleware for communication substrate. SIGAR API[1] was used for real-time hardware monitoring on PMs. The prototype is divided into three components: (i) Task Launcher; (ii) Scheduler Machine; (iii) Processing Machine. Each component can be mapped to a different machine. The Task Launcher is in charge of reading an input file containing the tasks to be processed by the system, sending them to the Scheduler Machine after that. Each line of the input file corresponds to a task, which presents a type and a time, in milliseconds, informing how long the system should wait before process the next line.

The Scheduler Machine is an implementation of the dispatcher, being responsible for scheduling the beforehand received tasks. This component reads an XML file informing the scheduler type (today we are supporting GetLB++, GetLB and RR), the verification period, the critical change percentage and the updating mode.

6. Evaluation Methodology

This section describes the environment used for the tests, starting by presenting the considered tasks in Table 1. There are three possible types of tasks, A, B and C, referring to balance, prepaid telephony, and purchasing transactions, respectively. Transactions data were collected from a real EFT company provider called GetNet. We are evaluating the tasks against three scheduling algorithms: GetLB++, GetLB, and RR. In addition, four different input files were used by the Task Launcher element, each one containing five thousand tasks to be processed. These files were generated with the intention of testing the framework and scheduling behaviors under the combination of different situations: (i) homogeneous and heterogeneous tasks; (ii) pure sequential without delay between tasks (when sending a task to the dispatcher) and waiting times between them. The heterogeneous tasks were generated randomly, while the waiting time is a quadratic random function ranging from 0 to 100 milliseconds. This quadratic function is pertinent to emulate

[1] https://support.hyperic.com/display/SIGAR/

the real functioning of an EFT company, that receives more transactions not in the start/end of the day, but close to noon or 18:00 hs.

Table 1: Characteristics of the tasks used in the tests

Type	Properties		External Systems			
	Class	Size (n)	Card Sys.	Cryptography	Fraud Prev.	Rech. Sys.
A	Balance	500	Yes	Yes	Yes	No
B	Prepaid	1000	Yes	Yes	No	Yes
C	Telephony Purchasing	2000	Yes	Yes	Yes	No

In addition we are also varying the verification period, percentage of critical change and update mode. These parameters are only valid in the context of GetLB and GetLB++. Here, a percentage of 100% indicates that the last measure must be at least two times greater when compared to the previous one to trigger a PM-switch communication. The configurations used to test the hardware where: (1) Critical with 15% of change and verification period of 500 ms; (2) Periodical with 15% of change and period of 10 ms; (3) Critical with 100% of change and period of 100 ms and (4) Periodical with 1% of change and period of 1 ms

6.1 Infrastructure Testbed

All tests were executed in the infrastructure of the Amazon Elastic Computer Cloud[2], where clusters consist of machines running Windows Server 2012 R2. We are working with two clusters, composing homogeneous and heterogeneous infrastructures. Both were formed by ten machines, where the homogeneous setting includes only machines named t2.micro (as labeled by Amazon) and a communication latency of 40 milliseconds. t2.micro is composed of a single-core CPU with 2.5 GHz, 1 GiB of memory and 20 GiB of storage (SDD). The dispatcher is also a t2.micro machine, being used to run the Task Launcher too. The heterogeneous cluster, in turn, has different hardware settings varying from t2.micro to c3.large machines.

7. Results

This section presents the obtained results, starting with performance and scalability tests, observing the time on scheduling procedures. After that, we developed two sections to accommodate performance and quality of mapping results over homogeneous and heterogeneous clusters.

7.1 Scheduling Performance and Preliminary Scalability Tests

We have prepared basic scalability test involving the creation of a homogeneous cluster. We are varying the number of PMs from 2 to 12 to test the overhead on the RR, GetLB, and GetLB++ scheduling algorithms. Figure 3 depicts then this context in different curves. As expected, Round Robin does not suffer major impacts as the number of machine increases. GetLB got the worst scheduling besides presenting a low scalability when enlarging the resource infrastructure. Clearly, GetLB++ outperforms GetLB in the scheduling performance, but we must take care with the current observations in the following way: the performance ratio between GetLB and GetLB++ is slightly lower when working with 12 machines. Technically, 6.02 and 5.59 are the performance rations for these two algorithms when analyzing 2 and 12 PMs, respectively.

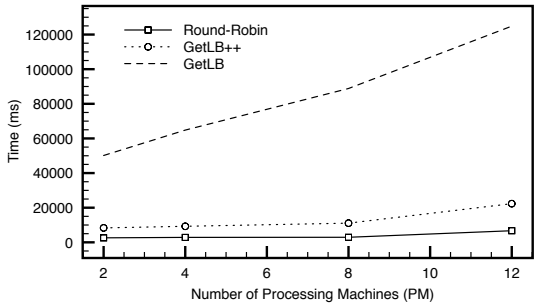

Fig. 3: Scheduling time when varying the number of PMs

7.2 Evaluation in Homogeneous Topology

This section presents the results when using a homogeneous cluster, but varying the type of the input workload. The graphics in this section show the load distribution among the PMs in the cluster. The values for these graphs were obtained by calculating the load level for each different type of tasks multiplied by the number of tasks of a particular type that were transferred to each machine.

7.2.1 Homogeneous Tasks

Figure 4 illustrates the load distribution of tasks among the PMs belonging to the homogeneous cluster. For this graph, it was used the test data of the first configuration of the update parameters, since there are no noticeable differences between this and the other settings. We observed a similar behavior on the three algorithms. Particularly, RR presents a completely uniform distribution of tasks to PMs because of considering, in this context, the duet resource and task as a homogeneous system.

7.2.2 Heterogeneous Tasks

Figure 5 shows a graph containing the load distribution of tasks in the homogeneous cluster for the three analyzed algorithms. Once again, the data used for the graph corresponds to the first configuration of the update parameters. It is noticeable that, while the Round Robin makes a homogeneous distribution, without considering that each type of task has special processing needs; the other two algorithms perform a fairer load distribution between the processing machines. Figure 5 shows that both GetLB and GetLB++

[2]http://aws.amazon.com/ec2

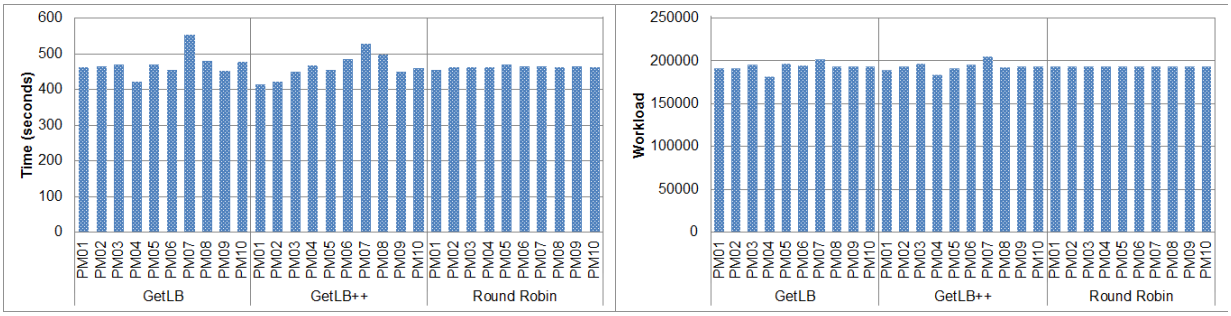

Fig. 4: Observing time and load distribution at each PM when running a homogeneous workload on the homogeneous cluster

Table 2: Processing times of a heterogeneous cluster with homogeneous workload. The column configuration follows the description in the Section 6.

Config.	Average Processing Time(ms) RR	GetLB	GetLB++
01	1640,73	1224.63	989.49
02	1640,73	1211.47	1005.40
03	1640,73	1229.62	971.20
04	1640,73	1356.21	1203.81

present a good load balancing since the homogeneous nodes (named in the graph as PMs) practically advance together in a horizontal line.

7.3 Evaluation in Heterogeneous Topology

This section presents the results obtained in the tests executed in the cluster composed by heterogeneous nodes. Here the results are presented in tables in order to show all four configurations values. Gains of 26% and 38% were obtained by GetLB and GetLB++ against the RR execution time over the configuration number two.

7.3.1 Homogeneous Tasks

Table 2 presents the results when handling homogeneous tasks. The standard deviations of these results were 32 ms for the average processing time. This expressive gain in performance was originated by the quality of the mapping provided by GetLB and GetLB++. We can observe that the higher the nodes capacity, the higher the load assigned to it. In addition, the use of heterogeneous resource turn more evident the differences between GetLB and GetLB++. Since GetLB++ does not try to evaluate a task against all PMs for scheduling purposes, but considers a decreasing sorted-list of previous mapped load to each PM, GetLB++ can provide both a better scheduling time and quality.

7.3.2 Heterogeneous Tasks

The scenario that puts together heterogeneous tasks and resources is responsible for the best results in favor of GetLB++. The obtained values are presented in Table 3, where the standard deviation of the average processing time is 41 ms. Gains up to 42% were obtained when comparing

Table 3: Processing times of a heterogeneous cluster with heterogeneous workload. The column configuration follows the description in the Section 6.

Config.	Average Processing Time(ms) RR	GetLB	GetLB++
01	3125.15	1870.12	1715.90
02	3125.15	1999.90	1790.62
03	3125.15	1989.59	1752.46
04	3125.15	1968.58	1706.07

GetLB++ and RR, and 11% in favor of GetLB++ when comparing its execution against the standard GetLB. The effectiveness of the RR scheduling calculus is totally ignored by the mapping provided by this algorithm. Analyzing the mapping of the load among the PMs we observed that the sequential method of RR leaves underloaded the more powerful resources.

7.4 Analysis and Discussion

As expected, RR was unbeatable when the term homogeneous is applicable for both resources and tasks. However, whenever a system component has a heterogeneous behavior (resources or task), Round Robin presented the worst processing time between the three analyzed algorithms. Homogeneous resources are responsible for a similar behavior between GetLB and GetLB++, presenting slightly better indexes for the last one. However, GetLB++ outperforms GetLB with heterogeneous resources both when considering uniform and non-uniform input workloads. The use of heterogeneous resources presented better results for GetLB++, which offers not only a faster scheduling calculus when compared to GetLB, but also a better mapping transactions-PMs for the following reasons:

8. Conclusion

According to the World Payments Report 2014[3], 70% of customers worldwide are expected to use mobile commerce in 2015 and more than 90% will likely be using online banking. So, this reality implies on performance challenges to EFT company providers, where the speed of an EFT

[3]https://www.worldpaymentsreport.com/

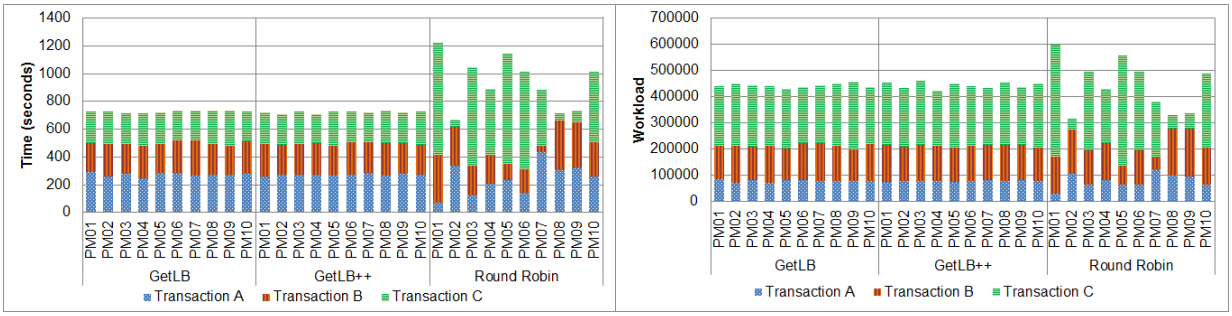

Fig. 5: Observing time and load distribution at each PM when running a heterogeneous workload on the homogeneous cluster

transaction can help on the client loyalty, while brings benefits to administrators who can handle more transactions per second, as well. In this context, this article addressed EFT performance through the GetLB++ proposal — an extension of GetLB especially focused on the scheduling procedure. GetLB++'s scientific contribution resides in the scheduling approach: contrary to GetLB, the number of PMs and the number of tasks mapped to each PM beforehand do not affect the scheduling performance. GetLB++ maintains a load-ranked decreasing-sorted list of resources and only takes the top position when arriving a new task. At each either hardware update at PMs perspective, conclusion of a task or dispatch of a task, this list is updated to reveal the most suitable quality scheduling.

According to the conducted tests, GetLB++ is in average 6.5 times faster than GetLB when performing the scheduling of a transaction, providing also in average a reduction in the total processing time by 11.78%. Furthermore, GetLB++ offers a more flexible framework, which allows the use of multiple scheduling algorithms, different PM-switch interaction approaches, and the processing of different kinds of tasks, not particularly EFT transactions. In this way, the contributions of GetLB++ are not limited to the context of transactional systems, but can also be extended for load balancing in e-commerce systems, cloud computing, and parallel programming. Heterogeneity at both resource and transaction levels were explored in the current paper to evaluate performance and scheduling quality. So, future work includes tests with resource dynamics and the use of notifications. In addition, we also intend to create a multi-cluster transactional environment considering different network latencies and bandwidths.

References

[1] D. R. Millen, C. Pinhanez, J. Kaye, S. C. S. Bianchi, and J. Vines, "Collaboration and social Computing in emerging financial services," in *Proceedings of the 18th ACM Conf. Companion on Computer Supported Cooperative Work & Social Comp.*, ser. CSCW'15 Companion. New York, NY, USA: ACM, 2015, pp. 309–312. [Online]. Available: http://doi.acm.org/10.1145/2685553.2685562

[2] B. Singh, R. Singh, and P. Singh Tanwar, "Electronic payment systems for online smart cards transaction system," *Int. Journal of Technology Research and Management*, vol. 1, no. 1, March 2014.

[3] C. Araujo, E. Sousa, P. Maciel, F. Chicout, and E. Andrade, "Performance modeling for evaluation and planning of electronic funds transfer systems with bursty arrival traffic," in *Intensive Applications and Services, 2009. INTENSIVE '09. First Int. Conf. on*. Valencia, Spain: IEEE, April 2009, pp. 65–70.

[4] I. Sbeity and M. Dbouk, "Software performance engineering using uml2san: Deadlock prediction of funds transfer," in *Computer Engineering Systems (ICCES), 2014 9th Int. Conf. on*, Dec 2014, pp. 318–323.

[5] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, pp. 1–53, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10922-014-9307-7

[6] J. Lewis, "The docline electronic funds transfer system (efts)," *Journal of Interlibrary Loan, Document Delivery & Electronic Reserve*, vol. 17, no. 3, pp. 75–83, 2007.

[7] R. Righi, C. A. da Costa, L. Gonzaga, Jr., K. Farias, A. L. Andrade, and L. Graebin, "Redesigning transaction load balancing on electronic funds transfer scenarios," in *Proceedings of the 29th Annual ACM Symposium on Applied Comp.*, ser. SAC '14. New York, NY, USA: ACM, 2014, pp. 775–777. [Online]. Available: http://doi.acm.org/10.1145/2554850.2555121

[8] R. Sastre, S. Bascon, and F. Herrero, "New electronic funds transfer services over ip," in *IEEE Electrotechnical Conf.*, 2006, pp. 733 –736.

[9] C. Vishik, A. Rajan, C. Ramming, D. Grawrock, and J. Walker, "Defining trust evidence: research directions," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW '11. New York, NY, USA: ACM, 2011, pp. 66:1–66:1.

[10] E. Derman, Y. Gecici, and A. Salah, "Short term face recognition for automatic teller machine (atm) users," in *Electronics, Computer and Comp. (ICECCO), 2013 Int. Conf. on*, Nov 2013, pp. 111–114.

[11] R. Priya, V. Tamilselvi, and G. Rameshkumar, "A novel algorithm for secure internet banking with finger print recognition," in *Embedded Systems (ICES), 2014 Int. Conf. on*, July 2014, pp. 104–109.

[12] E. Sousa, P. Maciel, and C. Araujo, "Performability evaluation of eft systems using expolinomial stochastic models," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE Int. Conf. on*. IEEE, Oct 2009, pp. 3328–3333.

[13] P. Desnoyers, T. Wood, P. Shenoy, R. Singh, S. Patil, and H. Vin, "Modellus: Automated modeling of complex internet data center applications," *ACM Trans. Web*, vol. 6, no. 2, pp. 8:1–8:29, 2012.

[14] H. Mcheick, Z. Mohammed, and A. Lakiss, "Evaluation of load balance algorithms," in *Software Engineering Research, Management and Applications (SERA), 2011 9th Int. Conf. on*. Baltimore, MD, USA: IEEE, Aug 2011, pp. 104–109.