Load balancing for distribuited databases access using a random heuristic algorithm

Algoritmo de balanceo de carga para Sistemas homogéneos distribuidos

R-D Sánchez Dams.[1], J-L. Simancas García[2]

[1]*Research Teacher GIACUC group assigned to Electronic Engineering Program at the Universidad de la Costa. E-mail: rsanchez5@cuc.edu.co.*
[2]*Research Teacher GIACUC group assigned to Electronic Engineering Program at the Universidad de la Costa. Email: jsimanca3@cuc.edu.co*

Recibido 29/10/13, aceptado 30/01/2014

Abstract

In this paper, the RanHeurist load balance algorithm is proposed, for homogeneous client-server distributed systems with a large number of requests. As a measure parameter the time response and the loss packets rate were used, comparing them with the Even Distribution for distributed systems. With RanHeuristic algorithm further information was processed in the same time period, improving delay rates in data reception, through a load distribution among two or more host processing, using randomness and a heuristic based on the best response time.

Keywords: Heuristics, random, load balancing, response time, distributed databases, concurrency access.

Resumen

En este artículo se propone el algoritmo RanHeurist de balanceo de carga para sistemas homogéneos distribuidos cliente servidor con gran cantidad de peticiones. Como parámetro de medición se utilizó el tiempo de respuesta y el números de paquetes perdidos, comparándolo con the Even Distribution for distributed systems. Se logró con RanHeurist procesar mayor información en un mismo periodo de tiempo mejorando los retrasos en la recepción de datos, utilizando una distribución de carga inteligente entre dos o más host de procesamiento, empleando aleatoriedad y una heurística basada en el mejor tiempo de respuesta.

Palabras clave: Heurístico, aleatorio, balanceo de carga, tiempo de respuesta, bases de datos distribuidas, acceso concurrente.

1. INTRODUCTION

Computer services are common nowadays to implement them, a widely archiecture is the client-server architecture, used for users' access to WEB services, or concurrent access to the same information system. In a client-server network, all the clients are connected to a server sending requests constantly, which are desirables to be answered in the shortest time [1, 2, 3].

On this architecture might happen that a large numbers of clients are performing requests to a server. If this doesn't have enough capacity to answer, this one might be saturated, by reducing the system performance, increasing response times, causing delays on information's delivery, by difficult the requests answers or causing that some client's requests won't be solved. To swap the requirement capacity, the load balancing emerges, which is responsible for divide the number of requests among two or more network host, in order to process higher information in the same period of time and thus optimizing the whole system performance [4, 5, 6].

Parallel to client's traffic must be remembered that management database systems are increasingly used to develop solutions in applications that handle large amounts of data, doing the sizes of the databases and the complexity of queries on it increase [6]. From this perspective, the load balancing shouldn't only be seen from the number of clients accessing the system also by the response time of it, because the more complex the query, the larger response time. In consequence, it should be keep in mind the host load and its setting, being relevant distribute the system workload in accordance to individual capacities and time responses of the host that constitute the system [4]. Moreover it's necessary make a load balancing capable to be optimized itself, establishing a proper load balancing.

According the above and [5], can be stated that load distribution is a key activity for an effective implementation, noticing that load balancing lies on the access speed increasing, improving the system accessibility, the failure tolerance, getting higher management and availability of service. To performance optimal load balancing, all the input traffic is distributed in several host, so the host performance is improve by using all the available resources, in other words, the load balancing ensures that an host won't be overloaded by a higher traffic influence [4].

Due to the importance of the topic, many people have done studies on this [7 - 12], to evoke there is the work made by Saneyasu Yamaguchi and Katsumi Maruyama whom implemented an autonomous load balancing system of distributed servers, using the active object model. This system consists of distributed servers where each server has a balancer that checks the load of the server and controls it, communicating with others load balancers. Additionally they implemented a Java library "CAPE" for Peer-to-Peer applications based on distributed active object systems [13].

Other work is done by J. Douglas Birdwell, Zhong Tang, and others, whom captured in their paper, the experimental results of a nonlinear dynamic system to equilibrate the load of a cluster of computer nodes used for parallel calculations, in the presence of delays and resource limitations. Also, "this model accounts for the trade - off between using processor resources to process tasks and the advantage of distributing the load evenly between the nodes to reduce overall processing time" [5].

Finally, the proposed solution is mentioned in [14], being the current work an independent alternative, using the same proof architecture for distribution and the same physical computational characteristics. These approaches differ

These two approaches differ in the proposed algorithms, which contrasts with the Even Distribution autonomously. In this paper two different load balance algorithm are compared, organized as follows: Section II describes all those fundamental concepts related to the approach made. Section III presents the heuristic algorithms and Even-Random Distribution proposed for solving the problem. In the IV section shows general features of the experiment setup. V in section details the results obtained as its analysis. Finally, in section VI concludes about this work and indicates the pending and / or consider for future work.

2. CONCEPTS

A. Client-Server Architecture and Distributed Systems

Client-server systems has its functionality split between server system and multiple client systems, i.e. this is a client making requests to another program (server) which gives answers. Depending on the type of architecture specifies the processing capacity is distributed between clients and servers to a greater or lesser extent. This type of architecture is of interest, mainly because provides organizational advantages, such as the centralization of information management and separation of responsibilities in systems, which facilitates and clarifies the system design [15].

A distributed system provides user access to the resources offered by the system. The access to a shared resource can be provided through data migration, calculations or processes mechanisms [16]. Could be say that, a distributed system is a loosely coupled processors collection interconnected by a communication network, that is, each has its own local memory and processors that communicate with each other through various communication lines, like high speed buses or networks [17].

Most distributed systems are based on the client-server model, which works as mentioned above. Due of this character sequential, in a distributed system can be implemented by RPC (remote procedure call) scheme, finding that most of tools for object distributed developing are based on RPC scheme [18]. The distributed programs [19] are another way of distribute responsibilities among host, using layers to concentrate functionality in each node, see Figure 1.

B. Distributed Databases DDB

Distributed database systems are a set of independent databases (ideally) sharing a common schema and coordinate the processing transactions that access remote data. Systems communicate each other through a communication network which handles the routing and connection strategies [20].

Database Distributed systems consist of a hosts set, each of which keeps a local database. Each host is able to process local transactions (transactions that only access data from the same host). Moreover, each host can participate in the performance of global transactions (access to data from multiple sites), and these are complied by communication among hosts is required [21].

The distributed databases implementation can be two types, homogeneous, in which all sites have a common databases schema and code systems, or heterogeneous, which the schema and code systems can be different [22]. According the setting for this work is homogeneous distributed databases.

C. Load Balancing and Availability.

A load balancer is basically a hardware device or software that is put in front of a set of hosts that serve an application, and assigns or balances the requests arriving from the clients to the hosts using some algorithm. The load balancing solutions allow splitting the tasks would have to endure a single machine in order to maximize the capabilities of data processing and task execution. Additionally, this solution allows availability; no host is vital part of the service offered [23]. In this way you can avoid suffering a service interruption due to a stop of one of the machines.

It's understood that load balancing in the computer context, consists on distribute some processing among hosts available for it. This concept applies to response time observed in the system [24]. On the other hand, availability can define as ability that a computer system has in order to serve users that are using it. This concept applies to loss generated in the system [25].

For ensuring that load balancing and system availability is successful it is important take into account the following characteristics.

Avoid an host overloading. In this way, we avoid that access peaks affect the application performance.
Manage the resources in smart way. Allows manage and optimize all the available resources, resulting a faster access and stability for clients.

D. Heuristics.

Heuristics is about explorative methods during problems solving, whose solutions are revealed by a progress evaluation achieved in a final outcome result. Heuristics characterize techniques by which the solving problems task result is improved. In computer systems, heuristic algorithms are used to obtain successful solutions without need to obtain exact solutions through calculus processes. However using heuristics it is likely incur in non-optimal results [26].

3. EVEN DISTRIBUTION AND RANDOM HEURISTIC ALGORITHMS PROPOSAL

In this section we present the two algorithms to compare, starting with the Odd-Even algorithm, which is a first approach to load balancing, where the queries made are distributed in an orderly and cyclical way. For example starts by sending a request to the remote host first, then the second, and so on until the last, and then repeated beginning again with the first. The following pseudocode represents described.

```
Host local recibe petición
If (host remoto actual == ultimo host remoto)
sendTo(primer host remoto)
Else
sendTo(siguiente host remoto)
EndIf
```

The second algorithm is called RanHeurist which had as design guideline introduces the least amount of processing, in order to generate lower computational load on the host that distributes the load, as recommended in [14]. To accomplish this comparison, simple instructions and assignments that consume fewer resources were chosen. Overall RanHeurist is based on identifying the best response time and associated remote host. This host is assigned a higher priority of occurrence, so that is most likely to be chosen. Due to this remote host is privileged to lower long it takes to perform the last query. Following is pseudocode and then a brief explanation of the algorithm:

```
Host local recibe petición
Aleatorio entre 0 y 1
If (numero aleatorio =< porcentaje privilegio me-
      jor host remoto)
      sendTo(mejor host remoto)
Else
      Aleatorio entre hosts remoto disponibles
      sendTo(host remoto aleatorio)
EndIF

Host local recibe respuesta remota
If (respuesta host remoto == mejor host remoto)
      Actualizo tiempo de mejor host
Else if (tiempo respuesta host remoto < tiempo
      mejor host remoto)
      Mejor host remoto ← host remoto
      Tiempo mejor host remoto ← tiempo respuesta
      host remoto
EndIf
```

As shown in the pseudocode, the algorithm makes a pitch for a pseudorandom number, if the result falls within the probability of choosing the best host, he is sent to it, otherwise it returns to make the launch random among all available remote hosts, holding everyone the same possibility. On the other hand when it is returned the result of the consultation made to last remote host, compares the processing time of this with you have registered as a better remote host. If obtained in the last query is better than we have, the best remote host is replaced.

According to the above heuristics is most likely to give a higher occurrence of probability to the last host with a better processing time, giving it the chance to pick a different remote host due to the random component of the algorithm. The random component to calculate time responses from other host, allowing find a better remote host and avoiding overload host with a better performance.

4. EXPERIMENTAL SETTINGS

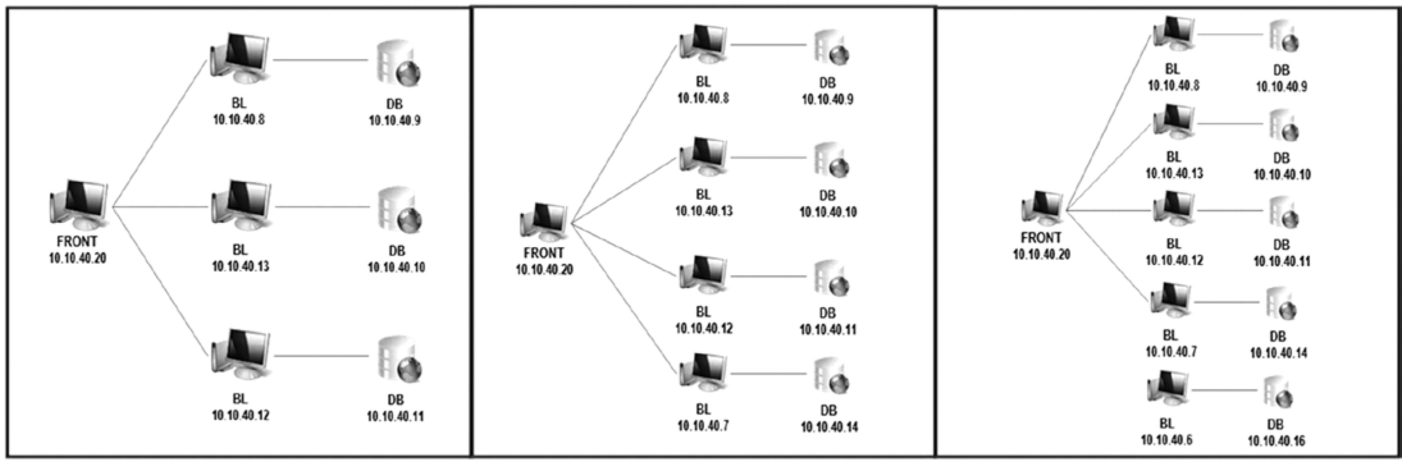
The metrics used in the tests of this study were twofold: response time and number of lost customers [27]. As mentioned earlier response times, measure load balancing and system availability measure lost. To implement the test, we used a transaction server client system. In this architecture, clients send the request to the server system, for he executes, or data sending results back to the clients so that they are displayed. Alternatives load balancing hardware or software, we used the logical approach, by installing the algorithms that run in back end in the host and were distributed computational burden.

The experiment was conducted in laboratory facilities at the Universidad del Norte in the city of Barranquilla, with the same configuration proposed in [14]. We used machines with Intel Core (TM) 2 CPU 6600 2.4 GHz, 2GB RAM, running NetBeans 6.9.1 and MySql Database over Windows XP, connected by an Ethernet switch with bandwidth of 100Mbps. The algorithms were implemented in Java, using a distributed system with distributed programs sokets communicated by using a layer for receiving requests (Front), one for the business logic, and a final layer for data. Moreover, in the sending algorithm, the metrics acquisition was implemented.

To perform the test, 16 computers were required distributed as follows: five (5) tester, responsible for sending requests to one (1) front load, balancing load to five (5) business logic, which in once they do so with five (5) databases. To verify controlled behavior under different conditions of the experiment was divided into three topologies using different numbers of business logic and data bases, as shown in Figure 1. Each tested separately.

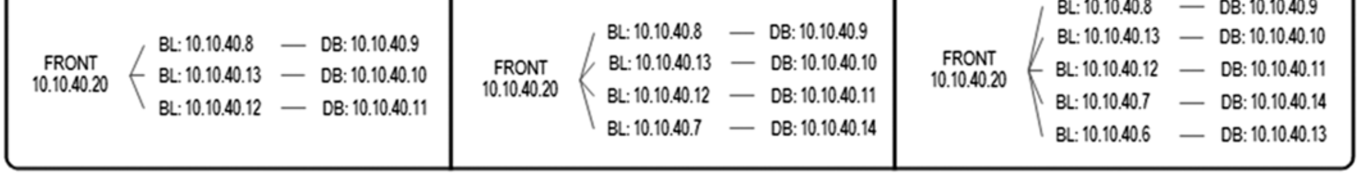
The figure above shows the logical architecture of the experiment; the A part, shows the experiment made with three business logical and its database, the B part, it shows made with four, and last, the C part with five logical business and its five database. Additionally, is observed the front host, which was the same during the entire test. The IP address setting used is showed in the figure 2.

Figure 1. Hosts architecture of the experiment
Figura 1. Arquitectura de Hosts del experimento



A. Three business logical B. Four business logical C. Five business logical

Figure 2. IP address used in the tests
Figura 2. Direccionamiento IP utilizado en las pruebas


FRONT 10.10.40.20
BL: 10.10.40.8 — DB: 10.10.40.9
BL: 10.10.40.13 — DB: 10.10.40.10
BL: 10.10.40.12 — DB: 10.10.40.11
FRONT 10.10.40.20
BL: 10.10.40.8 — DB: 10.10.40.9
BL: 10.10.40.13 — DB: 10.10.40.10
BL: 10.10.40.12 — DB: 10.10.40.11
BL: 10.10.40.7 — DB: 10.10.40.14
FRONT 10.10.40.20
BL: 10.10.40.8 — DB: 10.10.40.9
BL: 10.10.40.13 — DB: 10.10.40.10
BL: 10.10.40.12 — DB: 10.10.40.11
BL: 10.10.40.7 — DB: 10.10.40.14
BL: 10.10.40.6 — DB: 10.10.40.13

A. Three business logical B. Four business logical C. Five business logical

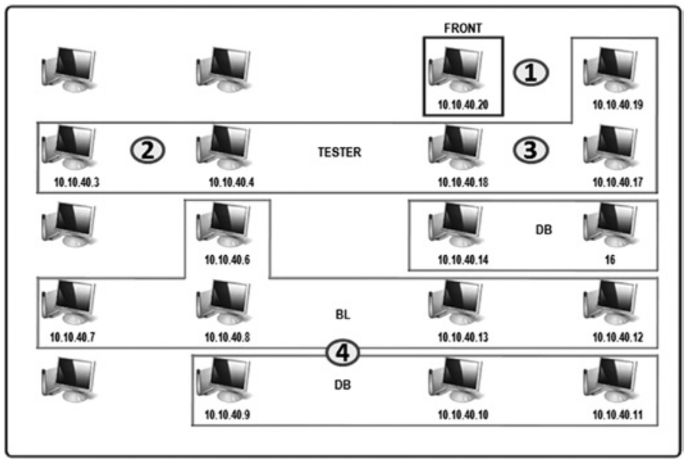
For each of the settings were made 300, 500 and 1000 request, in order to observe the load balancing algorithms' behaviors against a different numbers of requests made concurrently by tester clients, equitably distributing the requests made among them, as is showed in the following table.

Table 1. Numbers of requests made concurrently by tester clients
Tabla 1. Número de solicitudes realizadas concurrentemente por los tester clients

Requests total	Request by computer (5 testers Clients)
300	60
500	100
800	180
1000	200

Below is the physical location of the host within the laboratory network.

Figure 3. Computers physical location
Figura 3. Ubicación Física De Los Equipos



For conducting the test it took four (4) people which were distributed as illustrated. A person (number 1) in charge of the front and a tester, two (numbers 2 and 3) in charge of two testers, completing the five testers, people responsible for recording the information gathered from the test. The fourth person was in charge of the business logic and databases, with the work of being aware of unexpected failures and restart services before each test.

For each experiment configuration (number of request and amount of business logic and databases) was release a test. The release could be repeated a second time if for some reason the test was suspended for crashing the system, choosing the best of the results obtained between releases.

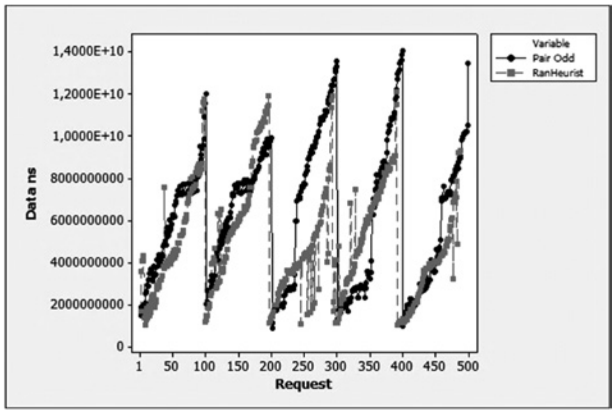
5. RESULTS

A. Drawbacks with loss metrics.

Although the metrics losses were taken into account in the tests, was not used in the analysis of results. The reason for losses excluding, lies in behavior observed during RanHeurist algorithm testing, which was done under different network conditions. The condition changes are attributed to the testing was performed one day after the Even Distribution, having to rewire the network again. In tests of the second day were observed anomalous behavior with request excessive losses the previous day were not presented in preliminary tests with RanHeurist, to check the correct operation of the algorithm performance, was monitored through messages on console, sensing that computers without reason and under low load request conditions lose connection among the front and the the business logic. Similarly again tested under the same conditions Even Distribution Algorithm getting excessive losses compared with the data (presented here) taken the day before. Despite the above tests were made under the same conditions of two attempts explained in the previous item.

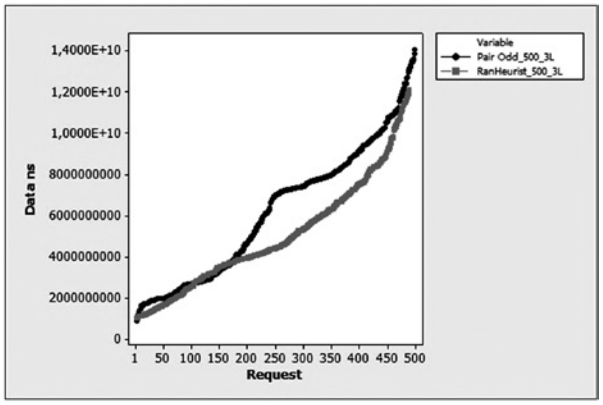
It was also observed that under equal conditions in the experiment, if the same test with moderate to high load was repeated several times, different results were obtained with more than 30% variation between tests, finding that the system sometimes simply crashed, worsening the second day situation as described.

Figure 4. Time series of even distribution; RanHeuristic, 500 requests, 3 business logical per tester
Figura 4. Series de tiempos de Even Distribution; RanHeurist, 500 Request, 3 lógicas del negocio, por tester



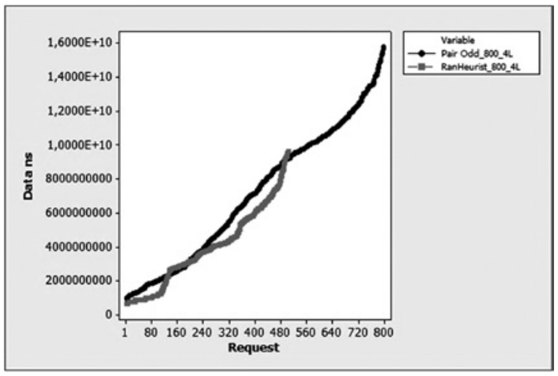
In the figure showed, the data are not organized, each lobe corresponding to data received by each of the five testers where the information was gathered. In this figure it is not clear which of the two algorithms has a better answer, in parts Even Distribution curve (black dots) is better than RanHeurist (red squares) and vice versa. Because the above is preferred to reorder data Ascending independent tester, simply observing the behavior of the system which is more suitable for analysis.

Figure 5. Time series of even distribution, RanHeuristic, 500 requests, 3 business logical
Figura 5. Series de tiempos de Even Distribution; RanHeurist, 500 Request, 3 lógicas del negocio

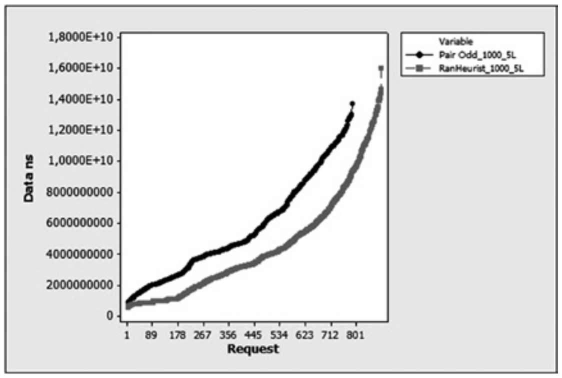


Below are a series of graphs representing the tests conducted in the experiment.

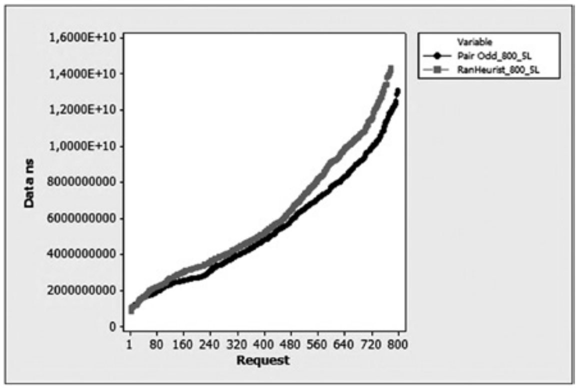
Figure 6. Time series representations of Even Distribution; RanHeurist
Figura 6. Series de Representativas de series de tiempos de Even Distribution; RanHeurist



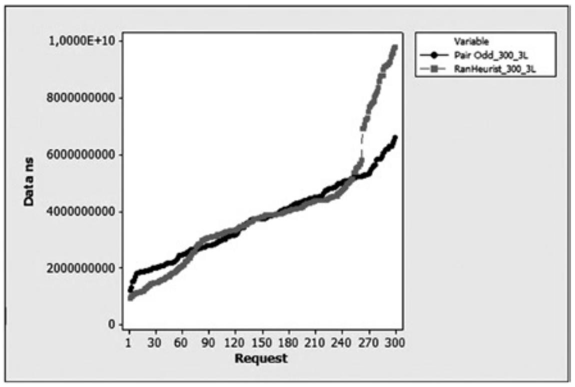
A.300 requests, 3 logics



B:800 requests, 4 logics



C. 1000 requests, 5 logics.



D. 800 requests, 5 logics

Analyzing visually the resulting curves is noted that RanHeuristic algorithm in most of graphs obtained has an improved performance over the entire test; however have asymptotic behavior at the end of each test release, with a rising slope steeper than that of Even Distribution. This asymptotic behavior is responsible for a significant deterioration in the average response time of the RanHeuristic algorithm. It also shows that at the beginning of a test or a request with few RanHeuristic algorithm behaves much like the Even Distribution, the latter being in these conditions and as noted in the graphs, a slightly better.

Below is a Table 2 which summarizes all tests in the experiment, observing in the last column the average response times. This table shows that in the twelve runs the RanHeurist algorithm releases 7 lower averages than the Even Distribution. Additionally raised a hypothesis test where the Even Distribution algorithm is better than RanHeurist, to which it was found that for all the statistical tests turned out to be less than the P-value (5%), so that for all cases, the initial hypothesis is rejected, finding that statistically RanHeurist algorithm gives better response times. Finally it is noteworthy that we observed the RanHeurist algorithm were remote host of business logic that process more requests over the rest.

Table 2. Averages with truncated data with lower number of requests
Tabla 2. Promedios con los datos truncados al menor número de request

Averages with truncated data with lower number of requests			
Number of logics	**Numbers of clients**	**Algorithm**	**Average time ns**
Three (3) logics	300	Pair - Odd	3746920937
		RanHeurist	3957345689
	500	Pair - Odd	6047718938
		RanHeurist	4976262397
	800	Pair - Odd	6922225030
		RanHeurist	7148378593
	1000	Pair - Odd	6305264984
		RanHeurist	6507325684
Four (4) logics	300	Pair - Odd	3717717077
		RanHeurist	3792965005
	500	Pair - Odd	4560593670
		RanHeurist	4316906615
	800	Pair - Odd	4563335257
		RanHeurist	3914769288
	1000	Pair - Odd	6346990202
		RanHeurist	3081528799
Five (5) logics	300	Pair - Odd	2977645743
		RanHeurist	2882842373
	500	Pair - Odd	4019191660
		RanHeurist	3926049928
	800	Pair - Odd	5245547878
		RanHeurist	6001844617
	1000	Pair - Odd	5626091481
		RanHeurist	3499735904

6. CONCLUSIONS

- Given the observed network conditions cannot draw definitive conclusions with respect to the metric of loss or that the experiments are comparable due to the change in conditions among the two experiments of the algorithms. Initially, and that observed in the tests is excluded that the use of the RanHeurist algorithm, is the predominant factor for excessive losses in test scores, given that by repeating the experiment in the same conditions with the Even Distribution, more requests were lost that tests originally made tests where data were taken for the present work. According to the above we propose the following hypothesis, that will be tested in future work done: The network played a dominant role in causing the observed communication problem, generating additional losses and an adverse environment for the experiment with RanHeurist, and introducing noise the tests conducted with the exposed load balancing algorithms.

- It is proposed as a future work, try RanHeurist with DEPRO algorithm and other load balancing proposals. For this experiment recommend debug in a controlled environment where the possible disturbances keeps restricted introduced in the experiment. These could be: to ensure that the test computers have the same software installed, without changing the roles of the host between tests, control the use of additional applications running on computers, monitor network traffic by restricting to only the test generated by isolating the test host with a router just for the experiment. Additionally it is recommended to establish additional protocols for the experiment and restart the computers before performing a test set for a specific algorithm. According to observations we conclude that the experiment has a large dispersion for the two metrics being raised from 20% to 30%, so it is recommended for future work to improve the conditions of the experiment according to the recommendations made.

- Another important aspect to take into account the decrease in the dispersion of the results is to focus on random queries that are performed in the experiment. To achieve this, two alternatives are posed; the first is to generate several releases for each test condition thus ensuring that the average consultation rate is similar for both tests. A second alternative is chose in a random way the queries to be used in each test, store them in an array and use exactly the same sequence of queries to the same conditions in the two algorithms. If this is not done in this manner is probably the same conditions for one of the two algorithms are working with queries to databases significantly more complex than the other. This could be a cause that has contributed to the observed inspired blockages in the system.

- The explanation for the better performance of Even Distribution algorithm, regarding RanHeuristic to the beginning of each test, or tests with low processing load is explained on the fact that in these conditions the remote hosts are not saturated, and because Distribution Even the algorithm is simpler uses less processing time which is reflected in slightly shorter answers. From this we conclude that the load balancing algorithm RanHeurist that react to a load, it makes greater improvements on performance if the remote hosts are not under stress. However, in cases where they undergo stress or remote hosts where these have different processing capacity, the RanHeurist algorithm has a better performance than Even Distribution, because this best advantage of remote hosts that have lower response time, despite the unfavorable conditions outlined above.

- So it is concluded that the RanHerustic about load balancing (response times) has a better performance than the Even Distribution, this claim is based on all the aspects evaluated (curve shapes, average response times, and hypothesis testing), the RanHerustic had a best performance than Even Distribution. Finally it is recommended that to improve operating conditions RanHeurist algorithm, taking into account the asymptotic behavior according as the system is saturated, the shooting should be defined according to the expected load on it, placing the maximum nominal load below the inflection point where the slope increases.

REFERENCES

[1] Edward Giovanni Arteaga Osorio, «Sistema Cliente Servidor para Visión de un Robot Móvil Usando Una Wireless Lan», Pontificia Universidad Católica del Peru, 2006.

[2] Ludwik Czaja, «On Deadlock and Fairness Decision Problems for Computations on Client-server Systems». 2011.

[3] Shinsuke Satake, Hiroshi Inai, Y Tsuyoshi Arai, «Effectiveness of Server Load Estimation by Using Requested File Size for Web Server Clusters», *Electronics and Communications in Japan*, vol. Vol. 94, n.º No. 2, 2011.

[4] Y.-T. Liu, T.-Y. Liang, C.-T. Huang, y C.-K. Shieh, «Memory resource considerations in the load balancing of software dsm systems», en *Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on*, 2003, pp. 71-78.

[5] Z. Tang, J. D. Birdwell, J. Chiasson, C. T. Abdallah, y M. Hayat, «Resource-constrained load balancing controller for a parallel database», *Control Systems Technology, IEEE Transactions on*, vol. 16, n.º 4, pp. 834-840, 2008.

[6] O. Dikenelli, M. O. Unalir, A. Ozerdim, y E. Ozkarahan, «A load balancing approach for parallel database machines», en *Parallel and Distributed Processing, 1995. Proceedings. Euromicro Workshop on*, 1995, pp. 51-58.

[7] J. L. Bosque, O. D. Robles, y L. Pastor, «Load balancing algorithms for CBIR environments», en *Computer Architectures for Machine Perception, 2003 IEEE International Workshop on*, 2003, p. 11 pp.-80.

[8] C. A. Yfoulis y A. Gounaris, «Online Load Balancing in Parallel Database Queries with Model Predictive Control», *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pp. 269-274, abr. 2012.

[9] L. Zhou, Y.-C. Wang, J.-L. Zhang, J. Wan, y Y.-J. Ren, «Optimize Block-Level Cloud Storage System with Load-Balance Strategy», en *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, 2012, pp. 2162-2167.

[10] A. Khanna, *Load balancing algorithm*. Google Patents, 2012.

[11] B. Janhavi, S. Surve, y S. Prabhu, «Comparison of load balancing algorithms in a grid», en *Data Storage and Data Engineering (DSDE), 2010 International Conference on*, 2010, pp. 20-23.

[12] M. Randles, D. Lamb, y A. Taleb-Bendiab, «A comparative study into distributed load balancing algorithms for cloud computing», en *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, 2010, pp. 551-556.

[13] S. Yamaguchi y K. Maruyama, «Autonomous load balance system for distributed servers using active objects», en *Database and Expert Systems Applications, 2001. Proceedings. 12th International Workshop on*, 2001, pp. 167-171.

[14] E. D. Nino, C. Tamara, y K. Gomez, «Load Balancing Optimization Using the Algorithm DEPRO in a Distributed Environment», en *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2012 Seventh International Conference on*, 2012, pp. 1-4.

[15] A. S. Tanenbaum y M. Van Steen, *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall, 2007.

[16] Bertocco, M, Ferraris, F., Offelli, C., y Parvis, M., «A client-server architecture for distributed measurement systems», presentado en Instrumentation and Measurement Technology Conference, 1998. IMTC/98. Conference Proceedings. IEEE, St. Paul, MN, 1998, vol. Vol. 1, pp. 67 - 72.

[17] Pianegiani, F., Macii, D., y Carbone, P., «Open distributed control and measurement system based on an abstract client-server architecture», presentado en Virtual and Intelligent Measurement Systems, 2002. VIMS '02. 2002 IEEE International Symposium on, 2002, pp. 63-67.

[18] Shaodong Ying y Shanan Zhu, «Remote laboratory based on client-server-controller architecture», presentado en Control, Automation, Robotics and Vision Conference, 2004. ICARCV 2004 8th, 2004, vol. Vol. 3, pp. 2194 - 2198.

[19] T. Elrad y N. Francez, «Decomposition of distributed programs into communication-closed layers», *Science of Computer Programming*, vol. 2, n.º 3, pp. 155-173, dic. 1982.

[20] Carlos Coronel, Steven Morris, y Peter Rob, *Bases de Datos, Diseño, Implementacion y Administracion*. Cengage Learning Editores, 2011.

[21] Laura Cruz Reyes, «Automatización del Diseño de la Fragmentación Vertical y Ubicación en Bases de Datos Distribuidas Usando Métodos Heurísticos y Exactos», Universidad Virtual Del Instituto Tecnológico Y De Estudios Superiores De Monterrey, 1999.

[22] Francisco Corbera Navas y Alejandro Delgado Gallego, «Modelos Avanzados de Bases de Datos». 01-abr-2008.

[23] Kermia O. y Sorel Y., «Load Balancing and Efficient Memory Usage for Homogeneous Distributed Real-Time Embedded Systems», presentado en Parallel Processing - Workshops, 2008. ICPP-W '08. International Conference on, Portland, OR, 2008, pp. 39 - 46.

[24] Li Jianxiang, Chuang Lin, y Fenglin Shi, «Availability Analysis of Web-Server Clusters with QoS-Aware Load Balancing», presentado en Computational Intelligence and Design (ISCID), 2010 International Symposium on, 2010, vol. Vol. 2, pp. 156 - 159.

[25] Chao-Tung Yang y Ko-Tzu Wang, «A VOD system on high-availability and load balancing Linux servers», presentado en Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on, Taipei, 2004, vol. Vol. 1, pp. 499 - 502.

[26] Carlos Rodríguez Ortiz, Abraham Duarte Muñoz, y Juan José Pantrigo Fernández, «Algoritmos heurísticos y metaheurísticos para el problema de localización de regeneradores.», Universidad Rey Juan Carlos, 2009.

[27] K. Abani, K. Akingbehin, y A. Shaout, «Fuzzy decision making for load balancing in a distributed system», en *Circuits and Systems, 1993., Proceedings of the 36th Midwest Symposium on*, 1993, pp. 500-502.