Explicit Use of Working-set Correlation for Load-balancing in Clustered Web Servers

Stoyan Garbatov and João Cachopo
INESC-ID Lisboa / Instituto Superior Técnico, Technical University of Lisbon
stoyangarbatov@gmail.com and joao.cachopo@ist.utl.pt

***Abstract*— This work presents a new load balancing policy for clustered web server systems. With this policy, every node in the cluster is responsible for dealing with a particular subset of request types. All requests are partitioned into disjoint groups, according to the domain data contained in their working sets. The composition of the working sets is established by using automatic data access pattern analysis and prediction techniques. Latent Dirichlet Allocation is employed as the partitioning algorithm to maximize the similarity (correlation) between the working sets of the requests placed in the same group. The working-set correlation is used explicitly with the aim of improving the data locality of all functionality that is to be executed on a given node, leading to more efficient use of computational resources and, ultimately, to increased performance. The work was validated on the TPC-W benchmark.**

Keywords-clustered web servers; load balance; locality awareness; Latent Dirichlet Allocation; scalability; performance.

I. INTRODUCTION

Scalability is a crucial property for many web systems. A system is defined as scalable if it is possible to change it (e.g., by adding more hardware) when the volume of requests increases, so that it maintains the same performance (in terms of throughput, response time, etc.). It takes some careful engineering to achieve good scalability. The solutions usually consist in designing the system in a particular way so that whenever some of the supporting (software or hardware) resources are upgraded, the system will be able to deal transparently with growing workloads without compromising its performance.

In the following, we use a classification of the existing approaches that is similar to the one presented by Cardellini et al. [5]. The simplest alternative for improving the performance of a web system is to upgrade the machine that is running the server to a machine with better specifications (e.g., processing units, disks, etc), which is referred to as hardware scale-up [7]. Unfortunately, this is a rather short-term and not very cost-effective approach because the increase of the workload, often driven by the increase of clients, far outpaces the hardware performance growth that is viable to be achieved for a single machine.

Many researchers concentrated their efforts on improving the performance of the server at the software level (software scale-up). Among these are improving the server's operating system [13, 14], developing more efficient web servers [16], and designing alternative request scheduling policies [6, 3]. Unfortunately, similarly to its hardware counterpart, improving the software performance of a single server is not a long-term solution to the web scalability issue.

Another class of approaches considers distributed systems composed of multiple servers. The main objective of this type of solutions is to spread the workload of the incoming requests among the existing server nodes, attempting to maximize the usefulness of the available resources. The policy for distributing the requests is usually performed by a single component called the load balancer. Any web system solution that achieves scalability and better overall performance by means of multiple server nodes is referred to as a scale-out approach [7].

These can be further refined into global and local scale-out solutions. The global scale-out approaches are characterized by having server nodes placed in geographically distinct locations, whilst a local scale-out has all nodes in a single network.

Our work falls into the category of the local scale-out approaches referred to as cluster-based web systems. These approaches are characterized by having a single entry point (the load-balancer), which is the only visible component from the client point-of-view. This is done for transparency purposes, so that the clients do not need to be aware of the potential multiplicity of entities that are effectively processing their requests, thereby avoiding all issues that such knowledge would entail.

The great majority of research in this field is concerned with the distribution of the workload in a uniform fashion among all server nodes. However, it has also been unequivocally demonstrated in [15], [1], and [8] that it is possible to achieve significant performance gains if the routing algorithm attempts to improve the data locality of server nodes when requests are being processed. In fact, it is well-known that to have good performance, a system should exploit and maximize its data locality.

In this work, we do not attempt to distribute the workload uniformly among server nodes. There are plenty of already existing solutions that can be combined into the solution that we propose here to give it uniform workload distribution properties. Instead, in this work we concentrate on the issue of improving the efficiency and performance of clustered dynamic content web systems, by developing a request routing algorithm that explicitly takes into account the existing correlation between the working sets of incoming requests, distributing them in a way that tries to maximize the data locality of operations performed at server nodes.

 ISBN: 978-1-61208-230-1

With the solution proposed here, every server node processes only a particular subset of request types: The requests are partitioned into disjoint groups, based on the contents of their working sets.

We identify the composition of the working set of each request type with the help of automatic data access pattern analysis and prediction routines. Then, we use the Latent Dirichlet Allocation partitioning algorithm to maximize the correlation between the working sets of all requests placed in a particular group. This is done to improve the data locality of all operations that are to be executed on a given server node, leading to a more efficient use of system resources and improved performance.

The contributions of this work are twofold. First, the explicit use of the correlation between request working sets is new and there is no previously existing work that makes request distribution decisions in a similar manner. Second, the system that we propose is entirely autonomous and self-sufficient in its operation,: from the analysis of the composition of the requests' working sets, up to the distribution of the incoming requests among operating server nodes, there is no point where human intervention is necessary in the decision making process.

The article is organized as follows. Section II discusses related work. Section III describes our new proposal in detail. Section IV presents the benchmark that we used to evaluate the new proposal and discusses the results obtained. Finally, Section V presents some concluding remarks.

II. RELATED WORK

Scalability is an essential property for web systems that are expected to deal with a large volume of traffic and extensive variations in the number of clients. Thus, it is not surprising the sheer volume of research that has been performed in this domain. Given the scope of our work, in this section we limit our discussion of related work to research regarding request distribution for clustered web systems. Cardellini et al. [5] performed a comprehensive study of locally clustered systems, whereas Amza et al. [2] evaluated transparent scaling approaches tailored for dynamic content systems.

Pai et al. [15] introduced the concept of locality-aware request distribution (LARD). The main idea behind this concept is that for a clustered web server system to have good scalability and operate efficiently, the load-balancing policy should take into account the content associated with incoming requests and redirect them so that the data locality of the server node responsible for processing them is improved. They use a hash function to partition the functionality provided by a given system (under the form of the types of requests that are available). The load-balancer uses this information to redirect requests so that every available server node is responsible for processing requests that belong only to a certain partition. This approach decreases the working-sets of all nodes to a portion of the system working-set, allowing for improved data locality and scalability. Pai et al. [15] performed simulations and validation on a working prototype, demonstrating that, through a locality-aware request distribution approach, it is possible to achieve significantly better scalability and performance than policies that do only uniform load distribution among nodes.

The main issue that can be pointed out in the work of Pai et al. [15] is that the partitioning of request types among server nodes is performed by a hash function. The fact that the partitioning mechanism is oblivious to the domain data accessed when processing requests does not give any guarantees as to the similarity of the types of requests redirected to a given node, in terms of the data necessary for their execution. Whereas such an approach may still lead to smaller-than-system working-sets at most nodes, it is far from optimal, because requests whose working-sets do not intersect can be placed in the same partition, degrading the overall locality of data. To maximize data locality, the working-sets of all requests placed in the same partition should overlap as much as possible, so that when a new request arrives at a node, it is more likely that most (if not all) of the data necessary for its successful completion is already available. This cannot be achieved without explicitly accounting for the relation between the functionality and the data needed for its execution.

Zhang et al. [19] present a detailed simulation study of AdaptLoad, which is a self-adjusting load-balancing policy that takes into account observed workload variations to tune its control parameters. With AdaptLoad, server nodes are responsible for dealing with requests that have similar sizes (e.g. processing times), seeking to minimize the overall job slowdown by separating the execution of differently sized tasks. The authors evaluate their approach against previously existing load-balancing solutions and demonstrate positive results, in particular when target systems are subject to highly dynamic load conditions.

The work of Amza et al. [1] presents a novel lazy replication technique, intended for scaling database back-ends of dynamic content site applications operating on top of computer clusters. This approach is referred to as conflict-aware scheduling and provides throughput scaling and one-copy serializability. This technique exploits the fact that, in the context of database clusters, there is a scheduler responsible for processing all incoming requests. By making use of information regarding the domain data accessed within transactions, Amza et al. [1] developed a conflict-aware scheduler that provides one-copy serializability, as well as reducing the rate at which conflicts occur. This is achieved by guiding incoming requests to nodes based on the data access patterns that are expected to occur during the execution of the associated transactions. Yet, there is a drawback of this approach that severely impairs its practicality: Programmers are responsible for providing information (under the form of a manually added tip) at the beginning of each transaction about which domain data is going to be accessed during its execution. Instead, we claim that the system should be capable of performing an automatic identification of the data access patterns that take place during transactions.

The work of Elnikety et al. [8] introduced a memory-aware load balancing method for dispatching transactions to replicas in systems employing replicated databases. The

 ISBN: 978-1-61208-230-1

algorithm uses information about the data manipulated in transactional contexts with the goal of assigning transactions to replicas so as to guarantee that all necessary data for their execution is in memory, thereby reducing disk I/O. For guiding the load balancing technique, the authors developed an auxiliary approach for estimating the volume and type of data manipulated during transactions. An additional contribution of their work is an optimization designated update filtering for decreasing the overheads due to the propagation of updates between replicas.

Zhong et al. [20] performed a study of the improvements that can be achieved by placing data when taking into account correlation information. They proposed a polynomial-complexity algorithm for calculating object placement that achieves a close to optimal solution with regards to minimizing communication costs. Further optimizations of the algorithm are indicated, by concentrating upon a small set of higher-importance objects. The approach is evaluated and demonstrated to produce significant reductions of communication overheads.

Given the existing related work described so far, we may draw some conclusions. Request distribution policies emphasizing data locality appear to offer much better scalability and performance gains, when compared against approaches concentrating only on load distribution. Uniform load distribution is still important and should be used to complement locality aware policies, but should not be the main goal. Despite all the good work done in the area of locality-aware load distribution, existing approaches have (at least) one of the following shortcomings:

- In their search for improved data locality, they do not take into account explicitly the data access patterns performed during execution of requests, either because they lack a proper analysis of data usage or because they expect that locality emerges “naturally” when requests are distributed among server nodes without taking into account the data manipulated in their contexts.
- The analysis of which data access patterns occur is performed manually.
- The clustering of requests/functionality among server nodes is performed manually.

The current state-of-the-art in the area of load distribution has ample room for improvements, which we explore with the solution proposed in this paper.

III. System Description

Our system is composed of three main modules – a data access pattern analysis module, an optimal clustering module, and a request distribution module.

The first module is responsible for identifying the composition of the working-sets associated with all types of requests that are executed by the system under consideration. This is achieved by analyzing and predicting the behavior of the target application in terms of the data access patterns that occur throughout the application’s execution contexts (such as methods and services). The analysis and prediction can be performed by means of one of three alternative stochastic model implementations, namely: Bayesian Updating [12], Markov Chains [10], and Criticality Analysis [9] (each of these references contains a thorough description and discussion of the implementation, functionality, and properties of the models). All three implementations provide highly accurate results that characterize the contents of the working set of any unit of functionality present in an application. The process of collecting behavioral data as input for this analysis is performed in an online fashion and incurs an average of 5% to 8% overhead in comparison with the original version of the target application performance. The relatively low overhead makes it feasible to gather this information even while the application is operating normally. It should be noted that all modifications necessary for the acquisition of the behavioral data are performed in a completely automated manner by the system presented here. The composition of the working-sets, as acquired by the first module, is then supplied as input to the optimal clustering module.

The second module, which has been described in more detail in [11], is responsible for identifying the optimal clustering of the target application's functionality (which, in this particular case, is characterized by the request types that the application provides) and domain data (working-set composition), based on the data access pattern behavior observed at runtime.

The algorithm used to perform the partitioning of application request types is the Latent Dirichlet Allocation (LDA) [4], which corresponds to the current state-of-the-art in multivariate clustering algorithms. By providing the composition of the working-sets associated with application request types as input to the LDA, the algorithm is capable of grouping the requests into several clusters. The partitioning is performed so that all members of a given cluster have the strongest possible (positive) correlation among themselves. For the current work, cluster elements are application request types, which are characterized by their working-sets. This leads to LDA generating as output clusters of request types, whose working-sets are very similar, because of their positive correlation.

However, there are several control parameters (among which is the number of clusters into which the elements should be grouped) that need to be provided as input to the clustering algorithm, and, thus, the only guarantee that LDA provides is that the cluster output composition maximizes the correlation between cluster elements, but only for the particular set of control parameters given. Unfortunately, there is no way to know, a priori, what are the control values that would lead to the best possible results. So, to identify the set of control parameters that leads to the highest quality results, we use the Silhouette technique [17]. Intuitively, good clusters have the property that cluster elements are (conceptually) close to each other and far from the elements of other clusters. The Silhouette technique captures this notion and provides an indicator value of how good a particular clustering is.

To find the optimal values of the control parameters for the LDA, our system calculates the average Silhouette values from several executions of the LDA algorithm for different configurations of the control parameters, within their valid

 ISBN: 978-1-61208-230-1

range of values. Once the Silhouette coefficients are available for all the evaluated scenarios, the values of the control parameters that produced the highest coefficient correspond to the optimal input scenario leading to the best possible clustering. The cluster results obtained after this step are provided as input to the request distribution module.

The request distribution module, which is where the new distribution policy is enforced, was implemented as a software switch, corresponding to Layer 7 (application) of the OSI protocol stack.

The switch acts as the sole entry point through which all incoming requests pass, before being distributed to a particular server node. It is the only visible component of the system, from the client point of view, making the multiplicity of application server nodes transparent to the clients.

Whenever a request arrives at the switch, it is examined, determining its type. The output provided by the optimal clustering module is consulted to determine the group to which this particular request belongs. Once this has been established, the request is forwarded to a server node that is responsible for processing requests of that particular group. After the request has been processed, the result is returned to the switch, and then forwarded back to the client.

As long as the target application supports having multiple instances of itself operating in parallel, without compromising application consistency, there is no need to make any modifications to the application for it to benefit from the request distribution functionality provided by the switch. The server nodes themselves are not aware of the presence of the switch, acting as if they were receiving their requests directly from the client.

The benefits of making sure that every server node deals only with request types that belong to a single group are the following. The effective working-set at server nodes should be significantly lower than the overall system working-set, not only because just a subset of all system functionality will be processed there, but also because that functionality was specifically selected with the purpose of maximizing the similarity of the composition of the working sets. This leads to more efficient use of computational resources (e.g. smaller memory footprints) and to better data locality, resulting ultimately in improved overall performance.

It can be argued that having all incoming requests and outgoing results going through the switch may be a performance and scalability bottleneck. However, we made this design decision to keep an implementation aspect of the proposed solution as simple as possible, because it is not in the request hand-off protocol that the main contribution of this work resides. Nevertheless, the switch implementation supports replication. So, multiple switches may be arranged in a hierarchical structure for the purpose of providing added performance or availability, at the cost of some added latency. This particular aspect of the work shall not be discussed any further.

IV. RESULTS

To validate the effectiveness of our approach, we used the TPC-W benchmark [18]. The TPC-W benchmark specifies an e-commerce workload that simulates the activities of a retail store website, where emulated users can browse and order products from the website.

The main evaluation metric used in our experiments is the number of web interactions per second (WIPS) that can be sustained by the system under test. The benchmark execution is characterized by a series of input parameters. The first of these indicates the type of workload, which varies the percentage of read and write operations that is to be simulated by the emulated browser (EB) clients. Three types of workload are considered here: Type1, with 95% read and 5% write operations; Type2, with 80% read and 20% write operations; and Type3, with 50% read and 50% write operations.

The analysis of the system was performed with the benchmark executing in Type2 mode. The same profiling results (from Type2) were employed for all 3 workload configurations (Type1, Type2, and Type3) in the performance testing phase.

The remaining input parameters for the benchmark were as follows: the number of EBs was fixed at 10; we used a ramp-up time of 300 seconds; the measurement was performed for 1200 seconds after the ramp-up time; the ramp-down time was 120 seconds; the number of book items in the database varied between 1k, 10k, and 100k; and the think time was set to 0, ensuring that the EBs do not wait before making a new request. All results were obtained as the average of 4 independent executions of the benchmark, with the same configurations. The EBs, the request distribution switch, and the benchmark servers were always running on the same physical machine.

All of the performance measurements were made with the benchmark running on a machine equipped with 2x Intel Xeon E5520 (a total of 8 physical cores with hyper-threading running at 2.26 GHz) and 24 GB of RAM. Its operating system was Ubuntu 10.04.3, and the JVM used was Java (TM) SE Runtime Environment (build 1.6.0 22-b04), Java HotSpot (TM) 64-Bit ServerVM (build 17.1-b03, mixed mode). The benchmark server nodes, as well as the request distribution switch, were run on top of instances of Apache Tomcat 6.0.24, with the options set to "-server -Xms64m -Xmx${heapSize}m -Xshare:off -XX: +UseConcMarkSweep GC -XX:+AggressiveOpts".

The performance results achieved when running the TPC-W benchmark with three different request distribution policies, and with three and four server nodes, shall be presented and discussed next.

The first policy corresponds to the new policy developed with this work. The second policy corresponds to an idealized locality-aware request distribution (LARD), where each server node is responsible for processing a subset of request types, and there are no intersections among the sets of request types assigned to different server nodes. This policy is idealized because it assumes prior knowledge of the composition of the workload, in terms of the relative proportions of incoming request types, as well as the average time that requests of a given type take to be processed. The policy attempts to achieve the most uniform load distribution possible, whilst keeping every server node dedicated to

 ISBN: 978-1-61208-230-1

processing a fixed subset of request types. The third policy employs a classic (unweighted) round-robin approach for distributing incoming requests among existing server nodes.

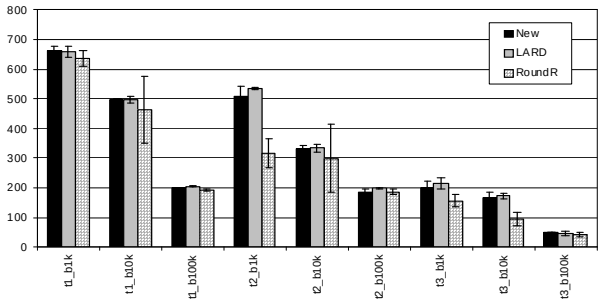

Figure 1. WIPs, 4 server nodes with 512MB heaps

The objective here is to demonstrate that it is possible to achieve improved throughput and efficiency in resource utilization by employing a request distribution policy that is explicitly aware of data locality, by taking into account the composition of the working-sets associated with request types. Given that, at this point of the work, there is no attempt to do a uniform load distribution, and that the conditions under which the system evaluation is performed (all server nodes share the computational resources of the same physical machine) do not allow for an objective and accurate evaluation of the load distribution among server nodes, no attempt shall be made at doing so. The evaluation metrics that we use here are only the overall system throughput and the efficiency in memory usage measured by the sizes of the effectively used heaps by the server nodes. A particular emphasis is given to the execution of the target application in sub-optimal memory availability conditions. These are expressed by running the server nodes with five different configurations for the JVM maximum heap size (Xmx): 512MB, 640MB, 768MB, 896MB, and 4096MB.

The throughput achieved by the three request distribution policies when the system is operating with 4 server nodes (where each is allowed to use up to 512MB of heap) can be seen in Figure 1. Each group of bars corresponds to a particular benchmark configuration, where t1, t2, and t3 indicate the type of workload, and b1k, b10k, and b100k indicate the size of the database used. By analyzing the results, it is possible to observe that the round-robin distribution is consistently outperformed by the other two policies, which display a rather similar throughput, across most configurations.

TABLE I. THROUGHPUT DIFFERENCE (%), 4 SERVER NODES

	512MB		640MB		768MB		896MB		4096MB	
	N/L	N/R	N/L	N/R	N/L	N/R	N/L	N/R	N/L	N/R
t1_b1k	0.6	4.2	-4.2	1.8	-7.1	5.1	-2.3	6.3	-1.4	2.9
t1_b10k	-0.1	6.8	-1.7	-4.7	-6.1	-4.5	-3.9	-1.8	-1.9	-5.1
t1_b100k	-2.3	3.8	0.0	3.7	-1.6	4.5	-2.6	3.2	-3.1	2.4
t2_b1k	-4.4	61.5	1.6	27.5	-1.9	18.5	9.3	19.8	1.6	19.1
t2_b10k	-1.3	10.1	0.5	3.8	-0.3	-8.5	0.0	-4.4	1.1	-5.5
t2_b100k	-7.4	-0.5	-3.4	3.8	0.2	6.4	-3.2	3.1	-1.8	0.2
t3_b1k	-6.7	28.8	-9.5	29.1	-9.9	31.1	-13.8	47.5	-10.8	26.2
t3_b10k	-3.1	76.7	8.4	47.2	2.6	48.3	-11.3	30.2	-9.7	26.1
t3_b100k	5.3	18.9	9.6	-42.6	16.2	-13.6	-6.1	-8.4	-8.4	-6.0
average	-2.2	23.4	0.1	7.7	-0.9	9.7	-3.8	10.6	-3.8	6.7

A thorough comparison of the performance gains achieved when using the New policy against LARD and Round-Robin, can be seen in Table I for 4 server nodes, and in Table II for 3 server nodes. The columns with "N/L" in the header contain throughput difference of New against LARD, calculated as $\left((T_{New} - T_{LARD})/T_{LARD}\right) \times 100$, %, whereas "N/R" is the comparative data of New versus Round-Robin.

TABLE II. THROUGHPUT DIFFERENCE (%), 3 SERVER NODES

	512MB		640MB		768MB		896MB		4096MB	
	N/L	N/R	N/L	N/R	N/L	N/R	N/L	N/R	N/L	N/R
t1_b1k	-2.07	5.13	3.95	28.89	-6.75	1.81	-2.65	2.46	-5.66	-4.10
t1_b10k	-1.06	-7.56	1.23	-7.91	6.18	-0.67	-10.11	-9.59	0.95	-5.99
t1_b100k	-2.53	3.37	-4.38	0.20	-3.08	0.97	-2.89	2.33	-1.30	3.95
t2_b1k	-7.40	42.26	-5.59	36.83	-1.50	11.53	-9.35	14.44	-6.14	11.39
t2_b10k	-6.39	-11.89	3.09	-3.89	-3.72	-8.78	4.17	-1.88	-7.76	-11.78
t2_b100k	-7.16	0.90	-3.03	1.78	-3.24	-0.47	-1.77	2.98	-3.56	2.03
t3_b1k	12.56	56.01	-5.07	40.55	14.21	74.91	3.32	63.02	0.46	46.36
t3_b10k	1.28	82.07	1.77	97.21	2.58	46.64	9.13	58.50	-11.55	35.50
t3_b100k	35.42	36.60	17.71	-8.35	-3.64	-31.28	7.65	-10.83	-6.64	-8.42
average	2.52	22.99	1.08	20.59	0.11	10.52	-0.28	13.49	-4.58	7.66

The New policy outperforms Round-Robin in 33 out of 45 (73%) configurations for 4 server nodes with an average of 11.62% better throughput, and in 29 out of 45 (64%) for 3 server nodes, with an average of 15.05%. The throughput achieved by using Round-Robin is the most inconsistent, across all configurations. This can be confirmed by observing Table III, where the uncertainty, expressed in terms of the covariance of the measurements, is displayed. The lower the value of the covariance of a given measurement, the higher the confidence in it - low covariance implies that the quantity being measured is unlikely to display values far from the mean.

TABLE III. MEASUREMENT UNCERTAINTY, 4 SERVER NODES

	512MB			640MB			768MB			896MB			4096MB		
	N	L	R	N	L	R	N	L	R	N	L	R	N	L	R
t1_b1k	0.03	0.04	0.053	0.00	0.08	0.17	0.07	0.03	0.05	0.01	0.04	0.02	0.01	0.02	0.04
t1_b10k	0.04	0.04	0.012	0.04	0.03	0.01	0.02	0.10	0.06	0.03	0.02	0.02	0.04	0.04	0.02
t1_b100k	0.01	0.02	0.009	0.00	0.03	0.01	0.01	0.01	0.02	0.00	0.01	0.01	0.01	0.02	0.01
t2_b1k	0.03	0.02	0.539	0.04	0.08	0.24	0.04	0.04	0.01	0.02	0.03	0.09	0.03	0.04	0.06
t2_b10k	0.03	0.04	0.051	0.01	0.04	0.07	0.03	0.07	0.02	0.05	0.06	0.02	0.03	0.05	0.03
t2_b100k	0.02	0.05	0.006	0.00	0.02	0.02	0.02	0.03	0.03	0.03	0.03	0.02	0.00	0.02	0.01
t3_b1k	0.20	0.07	0.058	0.11	0.02	0.06	0.13	0.09	0.11	0.09	0.14	0.16	0.21	0.08	0.10
t3_b10k	0.13	0.04	0.24	0.11	0.09	0.47	0.02	0.16	0.14	0.12	0.15	0.11	0.11	0.09	0.13
t3_b100k	0.21	0.03	0.237	0.02	0.09	0.32	0.06	0.17	0.15	0.06	0.08	0.20	0.08	0.12	0.13

The LARD policy outperforms the New approach in most configurations, by a small margin. On the average, the New policy provides 2.12% less throughput than LARD, for 4 server nodes, and 0.23% less throughput, for 3 nodes. Given that this particular LARD implementation is an idealized approach that makes use of perfect knowledge, a priori, about the target's system behavior, it is quite positive that the performance offered by the newly developed policy is so similar to an approach that would not be possible to achieve in practice.

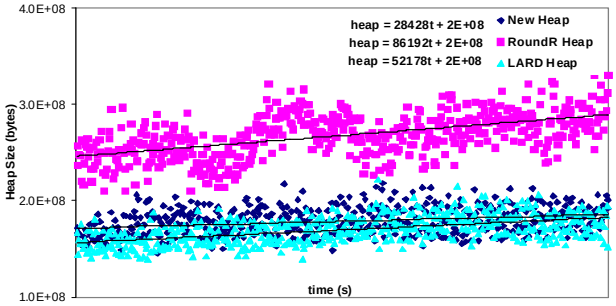

Figure 2. Heap size, t3, 10k books, 4 nodes, 640MB

Figure 2 contains the effective heap size usage achieved by the three policies, for a particular configuration of the benchmark, when 4 nodes are running with a maximum of

640MB of heap. The average heap size of New is 5.3% higher than LARD and 33.3% lower than Round-Robin. To identify the trends in memory usage we show also the result of a linear regression over the data collected. The gradients indicate that the heap growth rate of the New policy equals 0.33 of the Round-Robin and 0.54 of the LARD, resulting in the overall lowest growth in terms of heap.

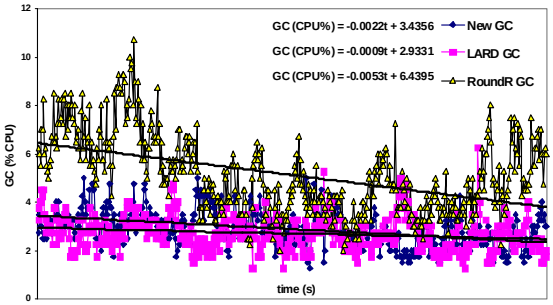

Figure 3. Garbage Collection (%CPU), t3, 10k books, 4 nodes, 640MB

Figure 3 shows the percentage of CPU spent performing garbage collection, for that same configuration of the benchmark. The average values are 2.9% for New, 2.7% for LARD and 5.1% for Round-Robin.

V. CONCLUSIONS

This work presented a new load balancing policy for clustered web server systems that seeks to maximize data locality by explicitly accounting for the correlation between the composition of the working sets of requests. That policy was applied to the TPC-W benchmark and evaluated against two alternative request distribution strategies. The newly developed approach provided significant performance gains under the form of increased throughput and improved efficiency in terms of memory usage, when compared against the alternative solutions.

ACKNOWLEDGMENT

This work was partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and by the Specific Targeted Research Project (STReP) Cloud-TM, which is co-financed by the European Commission through the contract no. 257784. The first author has been funded by the Portuguese FCT under contract SFRH/BD/64379/2009.

REFERENCES

[1] Amza, C., Cox, A. L. and Zwaenepoel, W., 2003, Conflict-aware scheduling for dynamic content applications, Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems, USENIX Association, Vol. 4, pp. 6-20.

[2] Amza, C., Cox, A. L. and Zwaenepoel, W., 2005, A comparative evaluation of transparent scaling techniques for dynamic content servers, Proceedings of the 21st International Conference on Data Engineering (ICDE 2005), IEEE, pp. 230-241.

[3] Bansal, N. and Harchol-Balter, M., 2001, Analysis of SRPT scheduling: Investigating unfairness, ACM.

[4] Blei, D., Ng, A. and Jordan, M., 2003, Latent dirichlet allocation, Journal of Machine Learning Research, 3 pp. 993-1022,

[5] Cardellini, V., Casalicchio, E., Colajanni, M. and Yu, P., 2002, The state of the art in locally distributed Web-server systems, ACM Computing Surveys (CSUR), 34 (2), pp. 263-311,

[6] Crovella, M., Frangioso, R. and Harchol-Balter, M., 1999, Connection scheduling in web servers, Proceedings of the 2nd conference on USENIX Symposium on Internet Technologies and Systems - Volume 2, Boulder, Colorado, USENIX Association, pp. 22-22.

[7] Devlin, B., Gray, J., Laing, B. and Spix, G., 1999, Scalability Terminology: Farms, Clones, Partitions, Packs, RACS and RAPS, http://www.scientificcommons.org/21762514,

[8] Elnikety, S., Dropsho, S. and Zwaenepoel, W., 2007, Tashkent+: Memory-aware load balancing and update filtering in replicated databases, ACM SIGOPS Operating Systems Review, 41 (3), pp. 399-412,

[9] Garbatov, S. and Cachopo, J., 2010, Importance Analysis for Predicting Data Access Behaviour in Object-Oriented Applications, Journal of Computer Science and Technologies, 14 (1), pp. 37-43, IEEE.

[10] Garbatov, S. and Cachopo, J., 2010, Predicting Data Access Patterns in Object-Oriented Applications Based on Markov Chains, Proceedings of the Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, pp. 465-470.

[11] Garbatov, S. and Cachopo, J., 2011, Optimal Functionality and Domain Data Clustering based on Latent Dirichlet Allocation, Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA 2011), Barcelona, Spain, ThinkMind, pp. 245-250.

[12] Garbatov, S., Cachopo, J. and Pereira, J., 2009, Data Access Pattern Analysis based on Bayesian Updating, Proceedings of the First Symposium of Informatics (INForum 2009), Lisbon, Paper 23.

[13] Hu, Y., Nanda, A. and Yang, Q., 1999, Measurement, analysis and performance improvement of the Apache web server, Proceedings of the Performance, Computing and Communications Conference, IEEE, pp. 261-267.

[14] Nahum, E., Barzilai, T. and Kandlur, D. D., 2002, Performance issues in WWW servers, IEEE/ACM Transactions on Networking, 10 (1), pp. 2-11,

[15] Pai, V., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W. and Nahum, E., 1998, Locality-aware request distribution in cluster-based network servers, Proceedings of the eighth international conference on Architectural support for programming languages and operating systems, San Jose, California, United States, ACM, pp. 205-216.

[16] Pai, V., Druschel, P. and Zwaenepoel, W., 1999, An Efficient and Portable Web Server, Proceedings of the 1999 USENIX Annual Technical Conference.

[17] Rousseeuw, P. J., 1987, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, Journal of computational and applied mathematics, 20 pp. 53-65,

[18] Smith, W. TPC-W: Benchmarking An Ecommerce Solution. Intel Corporation, 2000.

[19] Zhang, Q., Riska, A., Sun, W., Smirni, E. and Ciardo, G., 2005, Workload-aware load balancing for clustered web servers, IEEE Transactions on Parallel and Distributed Systems, 16 (3), pp. 219-233,

[20] Zhong, M., Shen, K. and Seiferas, J., 2008, Correlation-Aware Object Placement for Multi-Object Operations, Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems, IEEE Computer Society, pp. 512-521.

 ISBN: 978-1-61208-230-1