Autonomic request management algorithms for geographically distributed Internet-based systems

Mauro Andreolini, Sara Casolari, Michele Colajanni
Department of Information Engineering
University of Modena and Reggio Emilia
{mauro.andreolini, sara.casolari, michele.colajanni}@unimore.it

Abstract

Supporting Web-based services through geographical distributed clusters of servers is a common solution to the increasing volume and variability of modern traffic. These architectures pose interesting challenges to request management strategies where the most important goal is not to achieve maximum performance, but to guarantee stable and robust results. In this paper, we propose novel request management algorithms that are based on autonomic principles that is, on loose collaboration among the closest nodes and no knowledge about the global system state. Experimental evaluation shows that our autonomic-enhanced algorithms can guarantee robust performance in a variety of settings and reduce standard deviations of the response times with respect to existing request management algorithms.

1 Introduction

The Web has grown to a large scale in the number and variety of offered services and user population. The increasing degree of complexity behind Web-based services, possibly unpredictable user demands and flash crowds pose new challenges to the system architects. Server replication on a large scale has been increasingly applied to support any kind of Web-based services, because it is considered the best architecture to improve the quality of distributed services. Systems implementing replication at a geographical scale typically consists of multiple clusters of servers, where each of them hosts replicas of the Web resources. The design space for these architectures is huge in terms of strategies for system dimensioning, network positioning [13], replica placement [18, 25], data consistency [30].

The focus of this paper is on request management algorithms that should guarantee system stability and consequent user-perceived stability to respond to user expectation that Web-based services are available all the time and must respond soon despite any foreseeable augment in population. In particular, we want to evaluate the convenience of applying autonomic properties to request management strategies in these geographically distributed clusters. This innovative approach is motivated by some characteristics of the geographically distributed architectures that cannot be based on centralized solutions and strong interactions among the system components, and by the necessity of adapting the system behavior to continuous changes of unexpected events. Many existing solutions deal with traffic surges by over-provisioning the server system, but they waste resources and are unsuitable to face temporary and unpredictable flash crowds. On the other hand, the proposed management algorithms remove architecture bottlenecks by anticipating possible component overload and by gradually shifting portions of requests among the clusters. We integrate into the request management algorithms some fundamental autonomic concepts, such as decentralization of control, information collection and reflection, smooth adaptation to changing environment, loosely coupled collaboration with the closest nodes, and we demonstrate improvements in terms of stability and robustness of the user-perceived and system-perceived performance. To the best of our knowledge, this is the first paper that designs and investigates autonomic-enhanced algorithms for request management on a geographical scale. We exclude any centralized scheme [8] that can work well on a locally distributed scale, but not in a geographical scenario. We propose a trend-based activation scheme that is based just on local system information with no cooperation among other clusters. It is self-adaptable and guarantees best stability especially when it is enriched by a probabilistic approach to decide about redirection. On the other hand, existing strategies are founded on a threshold basis for activation/deactivation [14], that is not autonomic and risks to lead the system to unstable oscillations.

Another important novelty of this paper is its evaluation approach that is oriented to consider and prefer system stability and robustness instead of comparing algorithms on

the basis of optimal performance in short periods of times or for specific workloads.

The paper is organized as follows. Section 2 presents a typical model of a geographically distributed system and motivates the necessity of request redirection in order to anticipate critical load situations, and improve system stability and robustness of performance results. Section 3 discusses the design space of request management algorithms, including autonomic-enhanced policies. Section 4 analyzes main results of the experimental evaluation carried out for different workload scenarios. Section 5 discusses related work. Section 6 concludes the paper with some final remarks.

2 System model and motivation

Geographic replication has been employed by many kinds of Internet-based services, especially those managing large collections of data. Examples include content delivery networks [11, 22], peer-to-peer file sharing platforms [24, 19]. We focus on the Web context [8], although the overall approaches are applicable to other Internet-based architectures as well.

Systems implementing replication at a geographical scale typically consists of multiple clusters of servers that are placed in strategic Internet regions that is, Autonomous Systems (AS) that are very well connected with the other regions. The placement problem is out of the scope of this paper. For several economic, management and data consistency reasons, the number of clusters in the reality is in the order of some units: up to now, 4-5 sites are more common than 10-20 sites, although in the near future the most popular services will be hosted by more than ten clusters. Each cluster is typically characterized by a multi-tier architecture [26] and by a very large Internet connection (e.g., an OC-48, 2488 Mbit/s). In order to facilitate cooperation and information exchanges among the clusters, each cluster is well connected with the Internet regions hosting the other clusters (e.g., an OC-12, 622 Mbit/s). Without loss of generality, in this paper we assume that clusters are homogeneous in terms of server architectures and they hosts a replica of all the resources. A simplified model of the geographically distributed architecture is shown in Figure 1.

Whenever a client wishes to download a resource, the system typically selects a replica nearby and directs the request to the cluster from which a copy of that resource can be get. This first dispatching choice privileges the reduction of the transmission delay (that is, Internet proximity), but it does not take into account of the service time that a cluster needs to generate its response. A similar request management scheme is unable to respond to possible flood of requests that may reach a cluster, while some other clusters may be underutilized. Moreover, our experience suggests that Web performance perceived by the users is increasingly dominated by the server delays, especially when contacting busy Web sites [3]. In order to guarantee system stability and user-perceived stability, geographically replicated systems are often enriched by a second level dispatching where each cluster can handle inter-cluster redirections. Many specific techniques, including HTTP request redirection, URL rewriting, reverse proxies, exist to carry out request redirections. The problem is how to manage request distribution on a geographical scale with the goal of guaranteeing system stability and robustness. In Section 3 we present some autonomic-inspired strategies. Here, we motivate the necessity of requests redirection when the workload scenario is characterized by unpredictable load peaks, flash crowds and heterogeneous arrivals and non-uniform presence of the users in different Internet regions.

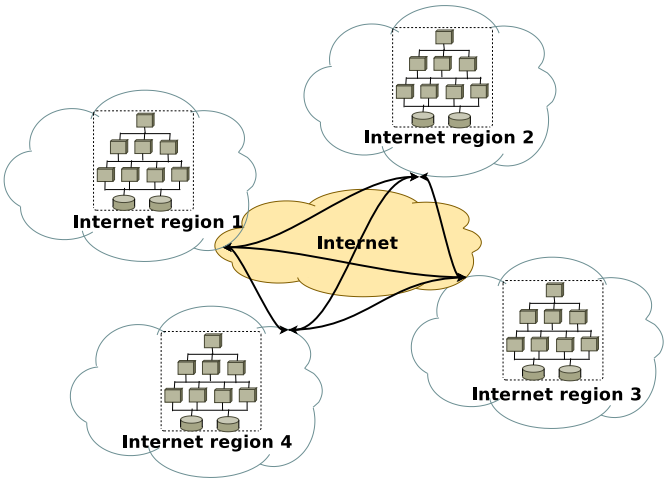

Figure 1. Model of a geographically distributed architecture supporting Web-based services

In a geographically distributed context, centralized solutions for redirection are inapplicable, but we see that fully decentralized schemes with no redirection are unsuitable as well, because they bring to severe load unbalance and consequent performance degradation. For example, let us consider the model in Figure 1. We assume that three clusters receive an unrealistic stable number of requests (Workload profile 1 in Figure2), and one cluster receives a realistic workload where the pattern of active users follows the profile presented in [4] (Workload profile 2 in Figure2). In the experiment lasting for 3600 seconds we consider that the clusters are subject to requests based on the TPC-W benchmark [29], and WAN effects between a client and a cluster, and among clusters are emulated through the netem package [15] using the realistic parameters that are described in Section 4.

In Figure 3, we report the mean Web page response times

over the experiment for Cluster 1 and 2, which are representative of the two sets of clusters (the mean is evaluated in intervals of 5 seconds). Cluster 1, which is subject to static load, preserves the mean response time around 1 second during the entire experiment, with a standard deviation of 183 msec. On the other hand, Cluster 2 shows a significant augment of the mean Web page response time (3.52 seconds) and, even worse, a severe fluctuation of the results resulting in a standard deviation of 2731 msec. (This is about 14 times higher than the standard deviation characterizing the response time of the Cluster 1 and, even worse, it may be amplified by the geographical context). Hence, we can confirm the intuition that, when the clusters are subject to realistic workloads, a fully decentralized scheme characterized by independent load management among the clusters does not help the system to guarantee stable and robust results. As centralized solutions are impracticable in a geographical context, in the next section we present some distributed request management algorithms that are enhanced by one or multiple autonomic principles.

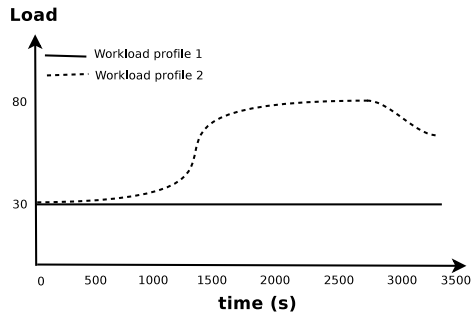

Figure 2. Patterns of the requests

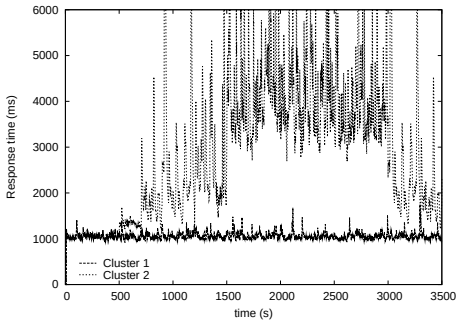

Figure 3. Response times in a fully decentralized system

3 Request management algorithms

Request management on a geographical scale is a critical process that requires decisions both in the design phase and at runtime. Moving load from one cluster to other(s) implies a tradeoff because each redirection consumes time and resources, and may increase the user perceived latency if the additional network time is not compensated by a reduced server time. From the system point of view, the costs of cooperative request management must be counterbalanced by the gains in terms of improved stability and robustness. In this section, we outline the request management design space, we discuss the alternatives that may lead to autonomic-enhanced algorithms, and we present a selection of algorithms characterized by non-autonomic, partially-autonomic or autonomic features.

3.1 Space of alternatives

Any request management algorithm includes a lot of alternatives at the level of the trigger mechanism, activation policy, selection policy, location policy, load mechanism, degree of cooperation.

- **Decision mechanism.** The decision mechanism specifies *who* decides about request management. The main alternatives are between a centralized or a distributed mechanism. In the former case, one component decides for all the incoming requests to which cluster they should be redirected. In the latter case, each Web cluster decides independently whether to accept or redirect a user request.

- **Activation/deactivation policy.** The activation policy determines *when* a Web cluster has to activate the redirection process and when it has to stop it. To preserve locality and integrity of the user sessions, we assume that requests may be redirected only at the beginning of a session; once assigned to a cluster, all the requests within that user session are served by the selected cluster.

- **Selection policy.** Once a cluster has decided to activate the redirection process, the selection policy determines *which* requests have to be redirected. The reassignment is non-preemptive that is, only requests that have not yet been served are eligible for redirection.

- **Location policy.** The location policy decides *where* the requests have to be redirected or, in other words, which cluster has to receive them.

The previous policies may be state-blind or state-aware in the sense that they take decisions on the basis of some

state information. In the state-aware instance, we can consider the *degree of information* that is, the number of clusters for state evaluation, and the *state information* that specifies the type of data that are considered for state evaluation, such as the monitored hardware/software resources, performance indexes, sampling frequency. In this paper, we consider the CPU utilization of the back-end nodes as the most representative information for the state evaluation. (This choice depends on the nature of the workload. Other workload models could lead to different choices.) In particular, during the experiments, we evaluate continuous state representations as the exponential weighted moving averages of the past 10 samples (EWMA_{10}) of the CPU utilization of the back-end nodes of the clusters [2].

3.2 Request management and autonomic properties

There are many possible alternatives for each request redirection choice: some are against the spirit of autonomic computing, others are may be considered partially or totally autonomic. We are interested to evaluate which of these choices can lead to autonomic-enhanced algorithms for request management that can improve robustness and stability of the geographically distributed Web system. Let us consider the most important properties behind an autonomic scheme.

- P_1**: decentralization of control.** The idea is to augment the system robustness and scalability by distributing the decisions among all the participating components (clusters). Removing the single point of failure characterizing a centralized decision scheme improves the fault tolerance too.
- P_2**: Information collection and reflection.** Each system component should be able to use its state information to acquire knowledge about its internal state. This knowledge can be used to decide between competing demands and to avoid possible bad decisions.
- P_3**: Adaptation to a changing environment.** Each system component should use state information about itself and other close components to dynamically adapt the request management process according to varying load conditions. One of the main goals is to keep all clusters away from overload situations (that is, *load sharing*, while load balance among the clusters is not pursued because considered unrealistic.
- P_4**: Loosely-coupled collaboration.** In order for a system to scale properly to large sizes while preserving robustness, any decision should be taken only in collaboration with a limited number of "neighbor" components. The main goals are to avoid communication overheads and to reduce stale information effects [9] that may easily occur on a geographical scale.

Based on these properties, we can classify the algorithms for request management into three categories: *non-autonomic* (no autonomic property is present), *partially-autonomic* (at least one autonomic property is included), and *autonomic* (all decisions are taken on the basis of some autonomic attribute). We present a significant selection of possible request dispatching algorithms by considering that an evaluation of all the alternatives is unfeasible in this paper. Table 1 summarizes the choices for each request management feature, and their impact on the autonomic properties.

Non autonomic algorithms This class of algorithms violates all the autonomic properties. A non autonomic algorithm may be based on a centralized trigger mechanism, activates/deactivates the redirection process state-blindly (or, at most, using very basic state information), does not adapt the redirection process in face of varying load conditions, and collaborates with the entire set of clusters. We consider two non autonomic algorithms.

The *CentrLL* algorithm is a classic example of least loaded assignment strategy across the whole set of nodes. It is based on a centralized decision mechanism that activates redirection when the load of the corresponding back-end node overcomes a threshold of 2/3 of its maximum capacity, and keeps on redirecting requests until the load falls below the same threshold. Requests are redirected to the cluster that has the lowest load at the back-end server. Every cluster in the system is eligible for redirection. The *CentrRand* algorithm is based on a centralized decision mechanism that is always active and redirects requests to a randomly chosen node.

Semi-autonomic algorithms (P_1)

In a geographically distributed context, the introduction of decentralized control is mandatory to remove the single point of failure and to favour scalability. We consider two algorithms based on P_1.

The *DistrLL* and *DistrRand* algorithms are two variations of *CentrLL* and *CentrRand* respectively, where the decision mechanism is distributed across all the participating clusters.

Semi-autonomic algorithms (P_1, P_2)

Property P_2 requires a specific load-aware activation process. For example, an activation based on one static threshold may cause unnecessary redirections, especially when the workload is extremely variable [23, 2]. One threshold allows to discern a binary (offloaded/overloaded) state, gives an instantaneous view about a component state, and does not help to understand whether the system is offloading or overloading. These lacks of knowledge contradict the property P_2. Moreover, continuous activation/deactivation may cause a system instability contradict-

Table 1. Design space of request management algorithms

Trigger mechanism	Activation/deactivation mechanism	Selection mechanism	Degree of collaboration	Autonomic properties
centralized	—	—	—	none
distributed	state blind	—	—	P_1
distributed	threshold-based	—	—	P_1
distributed	load trend	—	—	P_1, P_2
distributed	load trend	probabilistic	—	P_1, P_2, P_3
distributed	load trend	probabilistic	partial/local	P_1, P_2, P_3, P_4

ing the P_3 property. To enforce property P_2 and to augment the degree of knowledge associated to the server state, we propose that the redirection process be activated on the basis of the behavioral *trend* of the server load. The importance of considering the load trend instead of punctual or average values comes from the necessity of comparing the resource load not just in absolute terms, because it leads to wrong management decisions when the load measures are extremely variable. Figure 4 gives some qualitative examples of trends based on the past three load values. It is possible to discern a steady increase ($t = 4$) from a transient spike ($t = 3$). In this paper, we consider a trend-based activation mechanism, where redirection starts as soon as a steady increase is detected ($t = 4$). Deactivation occurs on all the other cases. The *DistrLLTrend* and *DistrRandTrend* algorithms enhance *DistrLL* and *DistrRand* by introducing an activation mechanism based on this behavioral trend.

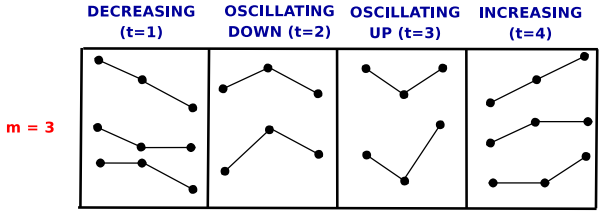

Figure 4. Trend load representation

Semi-autonomic algorithms (P_1, P_2, P_3)

The selection policy can be *state information blind*; this includes the straightforward solution of redirecting at *every* opportunity. However, a blind redirection that does not take into consideration the actual load conditions of the redirecting cluster may cause a huge amount of unnecessary redirections. Even worse, the lack of any feedback loop may introduce oscillations and system instability that would invalidate P_3. In order to favor adaptation to changing conditions and to avoid resource underutilization, it is also possible to perform a *probabilistic* selection strategy, in which only a percentage of the incoming requests is redirected, while the remaining ones are served locally. The probabilistic criterion chosen in this paper is simple yet effective: the percentage of requests to be redirected is uniformly proportional the last observed server load.

The *DistrLLTrendProb* and *DistrRandTrendProb* algorithms add probabilistic selection to their management decisions.

Autonomic algorithms (P_1, P_2, P_3, P_4)

All the previous strategies tend to operate with a specified degree of information about the load of the server nodes. In a *global* information context, a cluster should take decisions by considering the load of all the server nodes of the system. This approach conflicts against the most important autonomic properties, such as decentralization (P_1) and loose-coupling (P_4). We introduce a loose collaboration by limiting load information exchange to the two closest clusters. The *DistrLLTrendProbPart* and *DistrRandTrendProbPart* algorithms add this loose collaboration property.

4 Evaluation of the management algorithms

The ultimate goal of our analysis is to verify whether and to which extent the introduction of autonomic properties into request management algorithms for geographically distributed architectures yields a system that is robust with respect to the highly variable conditions of the Web. To this purpose, we measure the user-perceived stability and system-perceived stability of existing and proposed algorithms. We have carried out a large set of experiments for several workloads and system conditions.

4.1 Test-bed for performance evaluation

Each component of the geographically distributed architecture is a multi-layer cluster where the first layers execute the HTTP server and the application server, deployed through the Tomcat [28] servlet container; the back-end layer runs the MySQL [21] database server. We have added WAN delay among the clients and the clusters and among the clusters through the netem emulator [15]. Each cluster provides a single virtual IP address that corresponds to the address of the Web switch that schedules the requests and routes them among the other servers of the architecture. We

use a TPC-W like workload model [29] as the implementation in [7]. Client requests are generated through a set of *emulated browsers*, where each browser is implemented as a Java thread reproducing an entire user session with the Web site. We have experimented several workload scenarios, but in this paper we report the results for two workload models, *Profile1* and *Profile2*, with the following characteristics.

Profile1: it describes a TPC-W workload in the ideal case of number of clients that does not change during the experiment. (The shown results refer to a 30 emulated browsers). Although the number of clients is fixed, the number and type of requests reaching the system is subject to some variation because they follow the transition state diagram of the TPC-W model.

Profile2: it describes a realistic-like Web workload, obtained from Profile1, where the number of emulated browsers connecting to two of the participating clusters (Cluster 2 and 4) increases with the pattern presented in [4]. The length of the experiment is for 3600s.

4.2 User-perceived stability

For the purposes of this paper, we consider a system *stable* if it is able to keep the fluctuation of key performance indexes within specific operating conditions. Depending on the performance index, we can consider a user-perceived stability and a system-perceived stability. User-perceived stability refers to the variation of performance as it is observed by the users especially in terms of Web page response time that is, the time interval that occurs between the user click and the arrival at the client of the HTML skeleton and all embedded objects. Hence, a good measure of the user-perceived stability is the standard deviation of the Web page response times. To this purpose, we evaluate the sample standard deviation σ of all the Web page response times $\overrightarrow{R} = (r_1, \ldots, r_n)$, that is given by:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(r_i - \overline{r})^2} \qquad (1)$$

where $\overline{r}$ is the sample mean of $\overrightarrow{R}$. A large value of σ indicates that the data points are far from its mean, while a small value of σ indicates that they are clustered closely around the mean. This latter result is appreciated in terms of user-perceived stability.

Table 2 shows the user-perceived stability of all the considered algorithms for the workload profiles *Profile1* and *Profile2*. For every redirection algorithm shown in the first column, we report the average, 95-percentile and standard deviation of the Web page response time. Let us start with the lightest *Profile1* and focus on the non autonomic algorithms *CentrLL* and *CentrRand*. If we enrich these algorithms with all the autonomic properties P_1, P_2, P_3, P_4, we obtain the *DistrLLTrendProbPart* and *DistrRandTrendProbPart* policies. They improve user-perceived stability: 36% through least loaded location, 35% through random location. This result influences even the Web page response time that is substantially reduced in terms of average and 95-percentile values. However, while adding autonomic properties to random-based algorithms guarantees a valuable improvement in stability and response time, the same does not hold true for the algorithms based on least-loaded principles. If we introduce a decentralized control (property P_1) in the *CentrLL* policy and obtain the *DistrLL* algorithm, we achieve the best stability, with an improvement of user-perceived performance of about 39%.

Introducing more autonomic properties into *DistrLL* does not improve the overall stability of the system; in particular, adding load trend-based activation (property P_2) worsens stability. There is a motivation for this result. When the system is subject to a light workload and the activation threshold of *DistrLL* is set to 66% of the maximum cluster capacity, redirection is scarcely activated with a consequent reduction of network overheads due to redirection. If we use trend-based activation, the number of request redirections increases, even when not strictly necessary (for example, when the cluster is relatively offloaded, but its load is increasing). Consequently, the response times of *DistrLLTrend* are subject to higher variability. This effect can be mitigated by adding a probabilistic selection (property P_3) that is proposed by the *DistrLLTrendProb* algorithm. Probabilistic selection adapts the number of redirected requests as a function of the actual cluster load. In this case, even if an increasing load trend activates the redirection processes, a lower load index results in more requests being served locally with a reduction of the variability of the response times. While it is generally true that the adoption of autonomic properties improves stability, an integration that does not consider the characteristics of the architecture and its operating scenario (here, a light workload) can actually worsen the system performance.

Let us now to evaluate the performance of the request management algorithms when some clusters of the system are subject to a highly variable, although realistic, workload (*Profile2*). We can anticipate that the benefits of the autonomic properties are much more evident here.

If we introduce all the autonomic properties, we obtain an impressive improvement in terms of user-perceived stability with respect to non autonomic algorithms: about 495% for *CentrLL* versus *DistrLLTrendProbPart*; about 138% for *CentrRand* versus *DistrRandTrendProbPart*. It is worth noticing that the best improvements are achieved with respect to non autonomic algorithms that are based on least

Table 2. Stability of the considered algorithms

	Profile1				Profile2			
	avg	95perc	stddev	LBM	avg	95perc	stddev	LBM
CentrLL	1458.11	2356	569.71	2.28	2208.27	4492	5261.11	2.06
CentrRandom	1411.54	2264	542.72	2.30	1873.34	3919	1940.85	1.80
DistrLL	979.03	1664	346.61	2.03	1664.17	3546	2157.84	1.75
DistrRandom	1498.98	2244	497.5	2.15	1649.20	2626	877.41	1.94
DistrLLTrend	1204.14	2080	497.54	2.15	1511.14	3013	1265.11	1.68
DistrRandomTrend	1155.30	2063	480.64	2.14	1625.49	3171	3081.60	1.72
DistrLLTrendProb	1010.25	1835	387.67	2.05	1374.98	2899	1051.67	1.68
DistrRandomTrendProb	997.83	1807	375.93	2.05	1468.93	3137	1589.43	1.69
DistrLLTrendProbPart	984.98	1626	346.99	2.04	1312.85	2688	883.10	1.65
DistrRandomTrendProbPart	986.32	1633	349.99	2.03	1285.13	2524	808.99	1.59

loaded location. Under intensive workload conditions, these algorithms are often subject to load staleness [9] that might lead to server overload, resource exhaustion, and higher response times, as testified by the *CentrLL* algorithm. Introducing a distributed trigger mechanism (P_1) does not remove the load staleness problem, but it improves the scalability because the *DistrLL* algorithm redistributes requests only when the load is over the threshold. A trend-based activation (property P_2) leads to the *DistrLLTrend* algorithm that improves user-perceived stability because it introduces a feedback loop that activates the redirection process as soon as the cluster load increases (thus, preventing risks of overload). Unfortunately, the advantages of trend-based activation are paid by an increased network overhead due to the excessive augment of redirections. This effect is mitigated by the integration of the probabilistic selection (property P_3) that increases the percentage of local assignments when the cluster is not really overloaded.

If we consider the algorithms based on random location, the biggest gain is obtained by distributing the dispatcher controllers (see the *CentrRand* and the *DistrRand* algorithms in Table 2). In particular, the trend-based activation mechanism seems to influence very badly the user-perceived stability, although the system is more balanced internally. While trend-based activation actually strains to improve load balancing by favoring redirection, sometimes it is possible that the random location policy chooses an overloaded cluster, thus resulting in a high response time and consequent higher standard deviations of the response time.

Enforcing probabilistic selection (P_3) improves the request locality and reduces the risk of sending requests to an already overloaded cluster. A partial collaboration scheme among the clusters (property P_4) also seems to improve the user-perceived stability. From the previous considerations, we can conclude that, when the workload conditions are severe, a trend-based activation can improve load-aware location policies because it can offload a cluster quickly to the least loaded server. This does not seem to hold for load-blind policies, when there is a risk to choose an overloaded server.

We conclude our analysis with a first summary about our results for the considered algorithms and workloads:

- decentralization of control (P_1) does always bring benefits on user-perceived stability;
- trend-based activation (P_2) may act poorly on load-aware location algorithms under not severe workload conditions, and poorly on load-blind location algorithms under severe workload conditions;
- probabilistic selection (P_3) is always beneficial to user-perceived stability;
- loosely coupled collaboration (P_4) is beneficial to user-perceived stability.

4.3 System-perceived stability

System perceived stability refers to the variation of performance across the different system components, especially in terms of utilization. According to [12], a system is considered *robust* if it is able to guarantee stable performance despite possibly abrupt and intense variations in the operating conditions (e.g., flash crowds). We use a variation of the Kolmogorov-Smirnov index to quantify the robustness of a system in terms of the maximum distance between the cumulative distribution functions of Web page response times when the system is subject to different operating conditions.

System-perceived stability is captured also through the load balance metric [6] (LBM) that measures the degree of load balancing across the different nodes. For each cluster i

($i = 1, \ldots, c$), let us define a set of cluster load representations $\overrightarrow{L_i} = (l_{i_1}, \ldots, l_{i_m})$, computed in the $[1, \ldots, m]$ time interval, and let $pl_i = max(l_i \in \overrightarrow{L_i})$ be the highest load value. Then, LBM is defined as follows:

$$LBM = \frac{\sum\limits_{1 \leq i \leq m} pl_i}{(\sum\limits_{1 \leq i \leq m} \sum\limits_{1 \leq j \leq c} l_{j,i})/c} \quad (2)$$

The value of the LBM can range from 1 (system perfectly balanced) to the number of clusters c (system completely unbalanced). Table 2 reports the system-perceived stability of all the considered algorithms for the workload *Profile1* and *Profile2*. Many considerations on user-perceived stability also hold for system-perceived stability, with some exceptions that we detail below.

We recall that for *Profile1* autonomic-enhanced algorithms achieve a limited improvement of the system-perceived stability (about 10% using least loaded selection, 11% using random location). One reason for this limited improvement is that *Profile1* does not stress enough the system with cluster unbalance; consequently, the LBM values do not change much. However, the trend-based activation can be detrimental to load-blind location policies and beneficial to load-aware location policies.

If we consider the more interesting *Profile2*, we can observe that the system-perceived stability is improved by 24% and 13% when we enrich *CentrLL* and *CentrRR* with all the autonomic properties. Moreover, gradually adding these properties into all the algorithms based on least-loaded location seems to increment the internal stability. The same does not hold true for the random-based algorithms because when we integrate the trend-based activation (P_3), the LBM coefficient increases from 1.80 to 1.94. A deeper investigation reveals a nonuniform distribution of request redirections across the different clusters.

In Figure 5, we report the CPU utilization of the back-end node for the Cluster 1 and 2 of the prototype that is subject to the workload *Profile2*. All front-end nodes of the clusters implement the *DistrRandTrendProbPart* request management algorithm. At $t = 600s$, two clusters (including Cluster 2) begin to be subject to an augment of the load, while the other three clusters (including Cluster 1) receive the same amount of requests from the clients. The load increment activates the redirection process at the Cluster 2 that starts to redirect some requests to its closest clusters, that is, Cluster 1 and 3. We can appreciate that around $t = 1000s$, the load in all clusters is rather stable and balanced, although two clusters continue to receive a number of requests much higher than those received by the other three clusters.

We can conclude that decentralization of control (P_1) does usually improve system-perceived stability, unless the location policy is load-blind and does not put much effort into balancing. Moreover, the combination of autonomic properties, such as trend-based activation (P_2), probabilistic selection (P_3) and loosely coupled collaboration (P_4), is beneficial to system-perceived stability.

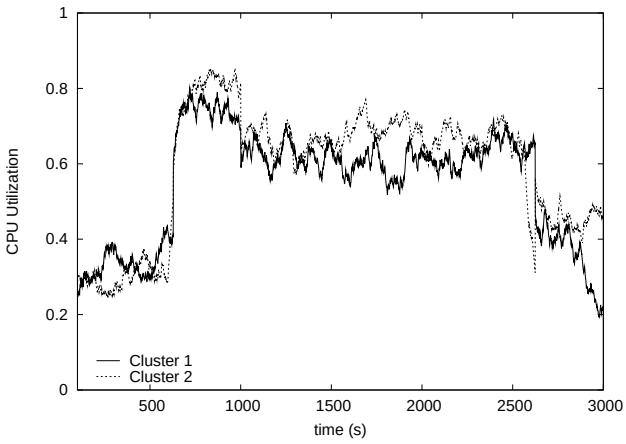

Figure 5. Internal cluster load

4.4 Robustness

We use as a measure of system robustness, the index defined in [12], that is a variation of the Kolmogorov-Smironov (K-S) statistics δ. We exercise our prototype under two different workload conditions: the reference workload condition *Profile1* that corresponds to a lightly loaded system, and a perturbed workload condition that provides a more severe scenario for the system. Then, we compute the cumulative distribution function of Web page response times $F(r) = P(R \leq r)$ for the reference workload and the cumulative distribution function of Web page response times $F^*(r) = P(R \leq r)$ for the perturbed workload. The δ statistics is computed as follows:

$$\delta_w = \sup_{-\infty < x < +\infty} (F(r) - F^*(r))\psi(r) \quad (3)$$

where $\psi(r) = -\ln(F(r)(1 - F(r))$ is a weight function that adjusts the index for the tail of the distribution. The Kolmogorov-Smirnov index measures the "distance" between the two CDFs. The closer the two functions, the less is the impact of the perturbation on Web page response times, hence the more robust is the system.

In Figure 6 we report the δ robustness index for some representative algorithms, when we apply four increasing workloads that span from *Profile1* to *Profile2*. The message from the figure is quite clear: enriching existing request redirection algorithms with autonomic-enhanced supports can consistently improve the robustness of the system. For example, the robustness increases by a factor of four

if we compare a centralized, non autonomic policy (*CentrRand*) and an autonomic (*DistrRandTrendProbPart*) algorithm. We observe that when we introduce more autonomic properties, the δ index grows slower, thus indicating a more robust system. If we compare the values of δ when the workload intensity is highest, we can conclude that the main contributions to robustness are achieved thanks to the following factors that are listed in terms of decreasing importance: loose cooperation (P_4, $\Delta\delta = 0.21$), decentralization (P_1, $\Delta\delta = 0.20$), probabilistic selection (P_3, $\Delta\delta = 0.10$)), and trend-based activation (P_2, $\Delta\delta = 0.09$).

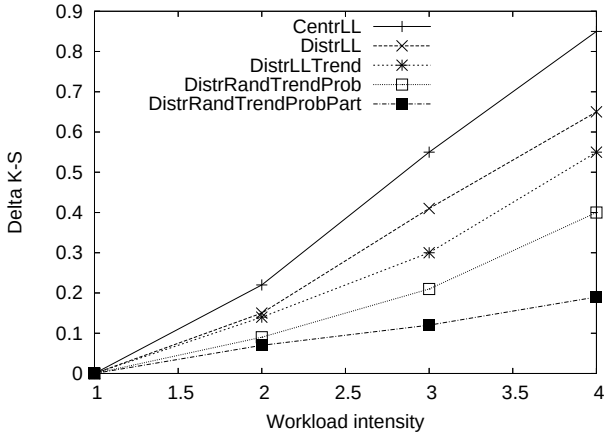

Figure 6. Robustness of the algorithms

5 Related work

The is a large literature on clusters, geographically distributed systems for Web-based services and related dispatching and load balancing algorithms. The initial idea comes from the observation that in a geographical scenario only distributed solutions are acceptable. However, this is the first paper that investigates whether autonomic-enhanced distributed algorithms for request management on a geographical scale may work and to which extent.

For example, existing strategies for redirecting load are founded on one or two thresholds for deciding about activation/deactivation [14]. An approach with fixed threshold(s) is clearly not autonomic and risks to lead the system to unstable oscillations. For this reason, we limit the space of investigation to a trend-based activation scheme that considers just local system information and requires no cooperation among other clusters. This is just an example because Section 3 contains other details and references.

Unlike the large majority of existing papers that evaluate performance through simulation models, we have carried a large set of experiments through a prototype of a geographically distributed Web-based system that is enriched with HTTP request redirection mechanisms, is subject to realistic workload and includes WAN effects.

We also observe that modern systems and solutions should be oriented to give more importance to system stability and robustness than to absolute performance, that in some sense is guaranteed almost always by top existing architectures. A similar evaluation approach applied to the geographical Web context is another original contribution of this paper. Measuring the robustness of a system is a widely studied topic, however previous approaches aim at specific problems such as job scheduling [5, 10, 17] that are quite different from the context considered here. Other approaches require a model of the system in order to make the associated mathematics tractable. For example, the papers in [20,27] need a Markov model that may be unfeasible for representing a geographically distributed system. Other definitions of robustness [1] are sensitive to some configuration parameters and, as a consequence, somewhat arbitrary. The δ robustness index used in this paper has been proposed in [12] to evaluate three systems (a job scheduler prototype, a streaming video server, a simulated distributed network service). We extend the robustness analysis to a geographically distributed Web-based prototype subject to realistic workloads. Furthermore, we analyze both the stability and the robustness of a system (the differences between stability and robustness can be found in [16]).

6 Conclusions

Highly popular Web sites require distribution infrastructures over geographical regions to scale to the ever increasing user demands. Unfortunately, these infrastructures cannot only rely solely on over-provisioning to peak demands, because they would be unable to face temporary and unpredictable flash crowds. In this paper, we have shown how the integration of fundamental autonomic concepts (such as decentralization of control, information collection and reflection, adaptation to changing environment, loosely coupled collaboration) into request management algorithms can lead to consistent improvement in terms of stability and robustness of the user-perceived and system-perceived performance. The proposed algorithms allow to reduce the communication overheads among the system components, remove architecture bottlenecks by anticipating possible component overload and by gradually shifting portions of requests among the clusters. As a consequence, the user perceives a reduced latency in terms of the 95-percentile of the Web page response time. All these goals are achieved through a novel, trend-based activation scheme that is based just on local system information with no cooperation among other clusters. It is self-adaptable and guarantees best stability especially when it is coupled with probabilistic selection based on a feedback control loop.

Another important contribution of this paper is the evaluation of system stability and robustness that must be preserved for the system to scale to larger sizes. The robustness of the performance obtained by some of the autonomic-enhanced algorithms is typically good (especially under heavy workload conditions), and seems to improve when adding autonomic properties.

References

[1] S. Ali. Measuring the robustness of a resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 15(5):630–641, may 2004.

[2] M. Andreolini, S. Casolari, and M. Colajanni. Models and framework for supporting run-time decisions in web-based systems. *ACM Tran. on the Web*, 2(3), 2008.

[3] P. Barford and M. E. Crovella. Critical path analysis of TCP trans. *IEEE Transactions on Networking*, 9(3):238–248, June 2001.

[4] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana. Predictability of Web server traffic congestion. In *Proc. of 10th Int'l Workshop of Web Content Caching and Distribution (WCW05)*, Sophia Antipolis, FR, Sep. 2005.

[5] L. Boloni and D. C. Marinescu. Robust scheduling of metaprograms. *Journal of Scheduling*, 5(5):395–412, 2002.

[6] R. B. Bunt, D. L. Eager, G. M. Oster, and C. L. Williamson. Achieving load balance and effective caching in clustered Web servers. In *Proc. of the 4th International Web Caching Workshop*, San Diego, CA, USA, USA, Apr. 1999.

[7] H. W. Cain, R. Rajwar, M. Marden, and M. H. Lipasti. An architectural evaluation of Java TPC-W. In *Proceedings. of the 7th International Symposium on High-Performance Computer Architecture (HPCA2001)*, Nuovo Leone, ME, Jan 2001.

[8] V. Cardellini, M. Colajanni, and P. S. Yu. Request Redirection Algorithms for Distributed Web Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):355–368, 2003.

[9] M. Dahlin. Interpreting stale load information. *IEEE Transactions on Parallel and Distributed Systems*, 11(10):1033–1047, Oct. 2000.

[10] A. Davenport, C. Gefflot, and J. Beck. Slack-based techniques for robust schedules. In *Proceedings of the 6th European Conference on Planning (ECP2001)*, pages 7–18, Sep. 2001.

[11] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *Internet Computing*, 6(5):50–58, 2002.

[12] D. England, J. Weissman, and J. Sadagopan. A new metric for robustness with application to job scheduling. In *Proc. of the 14th IEEE International Symposium on High Performance Distributed Computing (HDPC2005)*, pages 135–143, 2005.

[13] T. S. Eugene Ng and H. Zhang. A network positioning system for the internet. In *Proc. of the 2004 USENIX Annual Technical Conference*, 2004.

[14] P. Felber, T. Kaldewey, and S. Weiss. Proactive hot spot avoidance for Web server dependability. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS2004)*, pages 309–318, Oct. 2004.

[15] S. Hemminger. netem: Network emulator, 2004. – http://developer.osdl.org/shemminger/netem/.

[16] E. Jen, editor. *Stable or robust? What is the difference?*, pages 7–20. Oxford University Press, NY, USA, 2005.

[17] V. Leon, S. Wu, and R. Storer. Robustness measures and robust scheduling for job shops. *IEE Transactions*, 26(5):32–43, 2004.

[18] T. Loukopoulos, P. Lampsas, and I. Ahmad. Continuous replica placement schemes in distributed systems. In *Proceedings of the 19th annual international conference on Supercomputing*, 2005.

[19] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, pages 84–95, 2002.

[20] J. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, 29(8):720–731, 1980.

[21] MySQL database server, 2005. – http://www.mysql.com/.

[22] G. Pierre and M. van Steen. Globule: a collaborative content delivery network. *Communications Magazine*, 44(8):127–133, 2006.

[23] A. Sang and S. Li. A predictability analysis of network traffic. In *Proc. of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM2000)*, 2000.

[24] S. Saroiu, P. K. Gummadi, and G. S. A measurement study of peer-to-peer file sharing systems. In *In Proc. Multimedia Computing and Networking*, 2002.

[25] S. Sivasubramanian, G. Alonso, G. Pierre, and M. van Steen. Globedb: autonomic data replication for web applications. In *Proceedings of the 14th international conference on World Wide Web*, 2005.

[26] S. Sivasubramanian, G. Pierre, and M. Van Steen. Replication for web hosting systems. *ACM Computing surveys*, 36(3):291–334, Aug. 2004.

[27] R. Smith, K. Trivedi, and A. Ramesh. Performability analysis: measures, an algorithm, and a case study. *IEEE Transactions on Computers*, 37(4):406–417, 1988.

[28] The Tomcat Servlet Engine, 2005. – http://jakarta.apache.org/tomcat/.

[29] TPC-W transactional Web e-commerce benchmark, 2004. – http://www.tpc.org/tpcw/.

[30] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering web cache consistency. *ACM Trans. Interet Technol.*, 2(3):224–259, 2002.