Dynamic Pull-Based Load Balancing for Autonomic Servers

Remi Badonnel[1,2] and Mark Burgess[1]
[1]Oslo University College
St Olavs plass, 0130 Oslo, Norway
[2]LORIA - INRIA Lorraine, Nancy University
Campus Scientifique, 54500 Vandœuvre, France
Email: badonnel@loria.fr, burgess@iu.hio.no

***Abstract*— The growing autonomy of servers may significantly deteriorate the performance of traditional load-balancing strategies. Indeed, the authoritative decision belongs to the load-balancer, but the autonomous servers may reject the requests on their own convenience. We propose in this paper an original load-balancing strategy for transferring this authority from the load-balancer to the autonomous servers. We describe the underlying architecture and evaluate our solution based on a first set of experimentations.**

I. Introduction

A large number of methods [1] has been proposed in order to balance the workload among servers in an optimal manner. However, autonomic computing poses new challenges with regard to this issue. It favours autonomous servers that are weakly coupled rather than traditional hierarchical systems with strong couplings (based on an obligation model) and requires to redefine load-balancing strategies. Traditional load balancing is usually performed in a push-based (obligation) manner, which means the decision of whether a server should receive a request or not belongs exclusively to the load balancer. Autonomics sceptics often imagine that this kind of approach is fundamental to the idea of "control" and the idea of component autonomy stands in the way of proper resource sharing if servers will not do as they are told by an authoritative controller. We have already argued against this viewpoint [2] and show this belief to be erroneous below. In this paper we explore the arguments in favour of *pull-based* load balancing strategy (see figure 1) where autonomous servers can manage load at their own convenience. Each server knows its own capabilities and state more quickly and accurately than any external monitor, so it seems reasonable to explore the idea that it is the best judge of its own performance. As service requests are pulled from a load balancer, servers can decide to take on work depending not only on their available resources but also on its own internal parameters including their willingness to interact. The authoritative decision of the load balancer is therefore transferred to the autonomous servers.

The plan for the paper is as follows: we overview in Section 2 traditional push-based load balancing strategies such as DNS-based, dispatcher-based and server-based approaches.

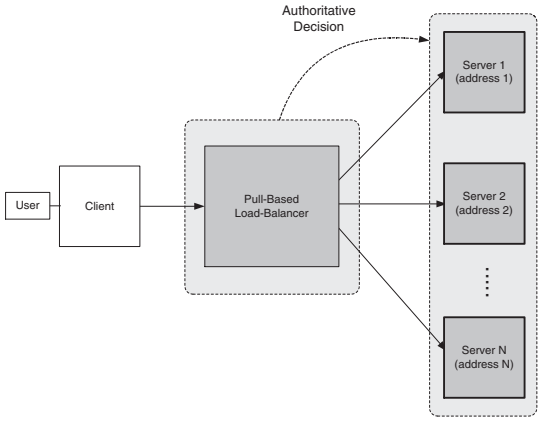

Fig. 1. Pull-Based Load-Balancing Strategy: the Authoritative Decision is Transferred from the Load Balancer to the Autonomous Servers.

We then define in Section 3 our pull-based load balancing scheme for autonomic servers, and describe the underlying architecture. We evaluate the performance of this solution through a set of experimentations in Section 4. Related work are detailed in Section 5. Finally, Section 6 concludes the paper with some pointers to future directions.

II. Load Balancing Schemes

A variety of algorithms and techniques address the issue of balancing load among servers, though much of the literature is quite old. For brevity, we consider only web servers in the following of the paper. Cardielli et al [1] proposed a classification of load balancing schemes into four categories:

- *Client-based scheme*: requests are load balanced by the client itself. This one dynamically selects to which web server to send the request. This requires the clients to be aware of several available web servers for a given service, and therefore shows a limited applicability.
- *DNS-based scheme*: the DNS server is responsible for load balancing requests through address mapping. When a DNS lookup is performed by the client, it decides to which physical servers the URL points to, by alternatively selecting IP addresses to be returned.

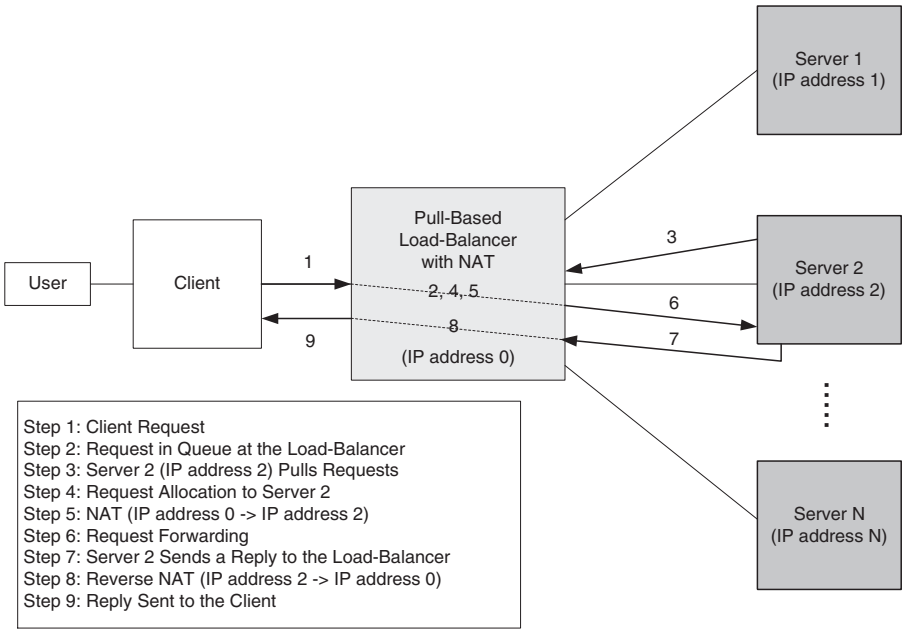

Fig. 2. Pull-Based load balancer with NAT

- *Server-based scheme*: web servers redirect requests to other servers when resources become overloaded. The redirection mechanism increases the latency perceived by the client.
- *Dispatcher-based scheme*: a dispatcher functions as a proxy. It hides the individual web servers and dynamically selects which of them will handle the request. The dispatcher-based load balancing can be implemented at level 2 or at level 3. This scheme is more appropriate for LAN environments.

Unsurprisingly, all these load balancing schemes are push-based [3]: a central entity (client, DNS server or dispatcher) directly pushes HTTP requests to the web servers. It is also the case for the server-based scheme, where requests are primarily assign by a DNS server.

III. Pull-Based load balancer

We propose in this paper to explore the benefits of a pull-based load balancer for distributing workload among autonomous servers. The key motivation is to make the load balancing strategy more flexible and adaptive with regards to the servers. This is compatible with the goals of autonomic computing. Autonomous servers interact only on a voluntary basis. Their autonomy contradicts the basic tenets of push-based load balancing.

With the push-based strategy, the decision of whether a host should receive a request or not, is taken by the load balancer. Some solutions permit the dispatcher to exploit state information transmitted (voluntarily and hopefully on time) by the web servers, but the push-based load balancer takes the final decision. In ref. [4], it was shown in actual hardware that this decision making could be a limitation on the dispatch rate. For instance, the least-connections algorithm [5] keeps track of the number of active connections each server currently has. The dispatcher then forwards requests (i.e. new connections) to the server with the fewest active connections. Another example consists of sending server-state information to the dispatcher using a dedicated protocol such as the Dynamic Feedback Protocol [6]. The dispatcher then determines based on that information which server will handle the request. This adds overhead to the process.

We consider the pull-based load balancer to be a dispatcher with a central queue. All of the requests from clients are kept by the dispatcher so that the web servers can pull requests from this queue at their own convenience i.e. whenever the server makes a voluntary decision to process a request. This includes whenever the server has free resources, but is not limited to that particular condition. A server may refuse to process requests for other reasons: when failures have been detected, when new components have to be installed and configured, when attacks have been detected and protection procedures have to be executed, or when the server simply does not want to (unable or unwilling to comply). In any event, the other servers adapt automatically to the best of their own ability and policy.

A. Step-by-Step Process

We will present how this strategy can be instantiated using a dispatcher with NAT (Network Address Translation) and will describe the process step by step. The NAT mechanism permits to translate the single public IP address of the load balancer to the (private) IP addresses of web servers. Figure 2 describes the main components of the architecture: client(s), pull-based load balancer, web servers. It depicts how they interact during the load balancing process.

We will detail the different steps of this process from the load balancer's point of view:

- *Client Requests*: the dispatcher receives multiple HTTP requests from clients via its public address. When a client sends such a request to the load balancer (step 1), the request is first stored into the central queue of the load balancer (step 2). The requests are usually processed according to a FCFS (First Come First Serve) queue scheduling discipline. The key difference with the push-based scheme is that the requests have to stay at the load balancer stage until autonomic web servers start to pull requests. The waiting time of client requests in the balancer queue can therefore be significantly higher than with the regular scheme.
- *Web Server Pulls*: each of the servers can pull requests from the dispatcher at any time. For instance in our scenario, Server 2 pulls requests (step 3) at its own convenience. If client requests have been issued and are waiting in the balancer central queue, the web server is dynamically served by the dispatcher. If this queue is empty or if several servers pull requests in a concurrent manner, server pulls are managed according to a predefined scheduling discipline (step 4). We can observe that the load balancing process has been reversed and that server pulls have to be managed by the dispatcher.
- *Dispatching of Requests to Web Servers*: the dispatcher uses the NAT mechanism to transfer a client request to a given web server. In our scenario, the load balancer public IP address is translated to the IP address of Server 2 (step 5). The client request is then forwarded to the web server (step 6). When the server has processed the request, the server sends the reply back to the dispatcher (step 7). The dispatcher translates the target IP address to the one of the client using Reverse NAT (step 8) and sends the reply to the client (step 9).

The NAT mechanism requires that the load balancer keeps state of each ongoing connection. This may contribute to a bottleneck effect [7] at the load balancer when the traffic rate is high. Three different strategies can usually be considered to send the reply to the client: (1) the bridge-path strategy when the dispatcher is implemented at level 2 and interacts as bridge, (2) the route-path strategy when the dispatcher is implemented at level 3 and interacts as an intermediate router, and (3) the direct-server-routing strategy when the reply is directly sent back by the server to the client.

IV. Experimental Results

We evaluated the behavior of our pull-based load balancing scheme through a set of experiments, which can be compared directly with earlier physical experiments performed in ref. [4], [5]. We performed the experiments with the open source suite for queuing network modeling and workload analysis [8] developed by the Performance Evaluation Lab at the Polytechnic University of Milan. This simulation tool can typically be used to experiment traditional push-based schemes. We extended it so that we can model the behavior of a pull-based load balancer with direct server return. We considered during the experiments a system composed of a load balancer L at the front office and a set of n servers $S = \{S_1, S_2, ..., S_n\}$ at the back office. The load balancer L implements either a push-based scheme or a pull-based scheme depending on the scenario. We assume that the arrival and completion process distributions follow a typical Poisson distribution in discrete time. Despite the controversy regarding the inter-arrival times, we use Poisson distributed arrivals, this allows a direct comparison with ref. [4] and we would not expect the choice to affect the broad conclusions of our results.

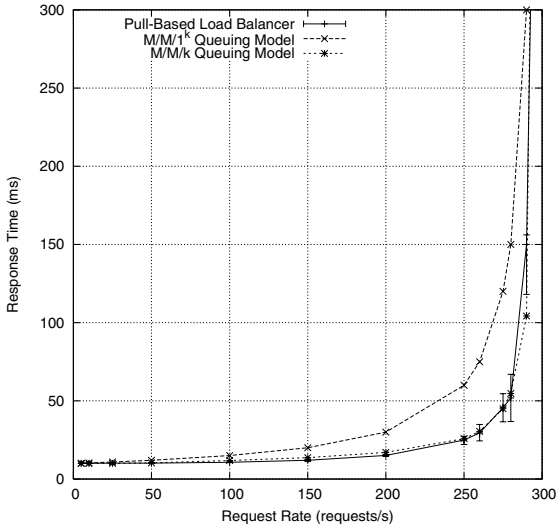

Fig. 3. Comparison of Response Times

In these experiments, we were interested in analyzing the response time of our pull-based solution. We modeled a system with three servers $\{S_1, S_2, S_3\}$ and measured the average response time obtained with the pull-based load balancer while varying the request rate from 5 to 300 requests per second. We consider here the natural notion of response time perceived by users, that is, the time interval between the instant of the submission of a request and the instant the corresponding reply arrives completely at the user. We compared these values with the response time provided by a push-based load balancer implementing the classic round-robin scheme where servers are taken each in turn without consideration of their current queue length or latency.

As previously mentioned, the autonomous servers are free to refuse requests on their own convenience: when specific management operations have to be performed (component installation or configuration, protection procedures), or when the server simply does not want to (unable or unwilling to comply). We assume an important restriction for these experiments: the decision of autonomous servers is only based on their maximal response rate. The autonomous servers start to reject requests when the request rate reaches the maximal response rate. Our results should therefore not come as a surprise if one considers the lessons of traditional queuing models. It is a well known result from queueing theory that a single

queue with k servers (denoted $M/M/k$) always out-performs k parallel pre-sorted queues with one server each (denoted $(M/M/1)^k$). In that context, intuition suggests that the pull-based model would behave like a strict $M/M/k$ queue, where as the push model will behave like an unbiased $(M/M/1)^k$ queue. This is confirmed by comparing the measurements to the theoretically computed values (see fig. 3). We clearly see that our pull-based load balancer produces better performance than the $(M/M/1)^k$ queuing model. Indeed, the pull-based strategy reduces the workload on each server by maintaining the requests on the load balancer. As a consequence, the system behaves in a very similar manner to a set of servers processing the same queue, and thus to the $M/M/k$ queuing model.

V. Related Work

Load balancing is a major issue for data centers. Techniques for balancing workload to back end servers are numerous [5]. Dynamic algorithms can exploit client-state and server-state information. The server-state information can be obtained in an implicit manner: the load balancer estimates the load on the servers by passively monitoring network parameters such as ongoing connections or by activating probes. In all of these approaches, the load balancing is performed in a push-based manner. We have already described in [4] a comparative study of these different load balancing algorithms used to distribute packets among a set of web servers in a push-based manner.

Server-state information can also be obtained in an explicit manner. For instance, the Dynamic Feedback Protocol (DFP) [6] permits a dispatcher to deploy agents directly on the back end servers. These agents periodically report relative weights to the load balancer in the form of load vectors. The load balancer then exploits the reported weights to select more efficient servers. This requires a substantial overhead and infrastructure. In distributed systems and grid computing, management platforms [9], [10] and service middlewares [11], [12] claim to dynamically align the allocation of resources to infrastructure and business requirements. These optimizations are often made possible thanks to server-state information collected by agents. In our approach, we show that there is no need for these feedback loops, completely autonomous behaviour suffices to solve the problem effectively by voluntary cooperation [2].

VI. Conclusions and Future Work

We have proposed a pull-based strategy for balancing workload between autonomic servers. The growing autonomy of servers contradicts traditional pull-based load-balancing, where the authoritative decision belongs exclusively to the load-balancer. Indeed, this authority is in conflict with an autonomous server's preference to make decisions by itself and to interacting on a purely voluntary basis. Moreover, if the balancer acts first, it has imperfect information about its opponent's condition and the coalition of balancer and servers can lose productivity through poor forwarding decisions. By considering a pull-based strategy, the authoritative decision is transferred from the load balancer to the servers themselves since this is where the important information is located. We have detailed an architectural solution for instantiating this strategy, and evaluated the performance through a set of experiments. When we restrict the decision of servers to a simple condition on a local resource, the experiments have showed that the system behaves in a very similar manner to a set of servers processing the same queue, and thus to the $M/M/k$ queuing model. Our future work will consist in refining the pull-based architecture and performing complementary experiments to measure other significative parameters such drop rates. We are also interested in evaluating the performance of voluntary dispatch in a large scale virtualization environment where these considerations are especially relevant.

Acknowledgment

This work was supported by the IST-EMANICS Network of Excellence (#26854).

References

[1] P. S. Yu, V. Cardellini, and M. Colajanni, "Dynamic Load Balancing on Web-Server Systems." *IEEE Internet Computing*, vol. 3, no. 3, June 1999.

[2] M. Burgess, "An Approach to Understanding Policy Based on Autonomy and Voluntary Cooperation." in *Proc. of 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'2005), Barcelona, Spain, October 24-26*, 2005, pp. 97–108.

[3] K. Gopalan, Y. Dong, and Z. Duan, "Push vs. Pull: Implications of Protocol Design on Controlling Unwanted Traffic." in *in Proc. of USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI'2005), Cambridge, MA, USA*, July 2005, pp. 25–30.

[4] M. Burgess and G. Undheim, "Predictable Scaling Behaviour in the Data Centre with Multiple Application Servers." in *Proc. of the 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'2006), Dublin, Ireland*, 2006, pp. 49–60.

[5] A. Berggren, "Presenting a Prototype for Pull Based Load Balancing of Web Servers." Oslo University College, Norway, May 2007.

[6] C. Kersey, "Dynamic Feedback Protocol (DFP)." http://dfp.berlios.de/draft-eck-dfp-00.txt, Aug. 2005, IETF Internet Draft.

[7] G. Jung, G. S. Swint, J. Parekh, C. Pu, and A. Sahai, "Detecting Bottleneck in *n*-Tier IT Applications Through Analysis." in *Proc. of the 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'2006), Dublin, Ireland*, 2006, pp. 149–160.

[8] M. Bertoli, G. Casale, and G. Serazzi, "An Overview of the JMT Queueing Network Simulator." Politecnico di Milano - DEI, Tech. Rep. TR 2007.2, 2007.

[9] S. Singhal, M. F. Arlitt, D. Beyer, S. Graupner, V. Machiraju, J. Pruyne, J. Rolia, A. Sahai, C. A. Santos, J. Ward, and X. Zhu, "Quartermaster - a Resource Utility System." in *Proc. of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM'2005), Nice, France*, 2005, pp. 265–278.

[10] M. Gohner, M. Waldburger, F. Gubler, G. D. Rodosek, and B. Stiller, "An Accounting Model for Dynamic Virtual Organizations." in *Proc. of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007), Rio de Janeiro, Brazil*, May 2007, pp. 241–248.

[11] E. Magaña, L. Lefèvre, and J. Serrat, "Autonomic Management Architecture for Flexible Grid Services Deployment Based on Policies." in *Proc. of the 20th International Conference on Architecture of Computing Systems (ARCS'2007), Zurich, Switzerland*, Mar. 2007, pp. 157–170.

[12] C. Adam, R. Stadler, C. Tang, M. Steinder, and M. Spreitzer, "A Service Middleware that Scales in System Size and Applications." in *Proc. of 10th IFIP/IEEE International Symposium on Integrated Management (IM'2007)*, Munich, Germany, May 2007.