Teaching High-Performance Service in a Cluster Computing Course

Pedro López[a,*], Elvira Baydal[a]

[a] *DISCA Department*
Universitat Politècnica de València
Camino de Vera, 14
46022 Valencia (SPAIN)

Abstract

Most courses on cluster computing in graduate and postgraduate studies are focused on parallel programming and high-performance/high-throughput computing. This is the typical usage of clusters in academia and research centres. However, nowadays, many companies are providing web, mail and, in general, Internet services using computer clusters. These services require a different "cluster flavour": high-performance service and high availability. Despite the fact that computer clusters for each environment demand a different configuration, most university cluster computing courses keep focusing only on high-performance computing, ignoring other possibilities. In this paper, we propose several teaching strategies for a course on cluster computing that could fill this gap. The content developed here would be taught as a part of the course. The subject shows several strategies about how to configure, test and evaluate a high-availability/load-balanced Internet server.

*Corresponding author
Email addresses: plopez@disca.upv.es (Pedro López), elvira@disca.upv.es (Elvira Baydal)

A virtualization-based platform is used to build a cluster prototype, using Linux as its operating system. Evaluation of the course shows that students knowledge and skills on the subject are improved at the end of the course. On the other hand, regarding the teaching methodology, the results obtained in the yearly survey of the University confirm student satisfaction.
Keywords: Linux clusters, computer engineering education,
high-performance service, load balancing, high-availability

---

1. Introduction

A cluster is a type of parallel or distributed processing system which consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource [7]. Although the idea of using clusters to improve system reliability dates back to the 60s [25], its usage to improve system performance is more recent [6].

The excellent price-performance ratio of clusters has led to their widespread usage. They fit the needs of multiple environments, from small servers to Small and Medium Enterprises (SME) and large Internet servers. Popular Internet services are provided by large scale clusters, usually located in different places to provide resilience against natural disasters. On the other hand, the largest supercomputers are also based on computer clusters [32].

Computer engineering curricula may include some subjects on clusters [15]. Among the topics to be taught, we can mention system architecture, networking, operating systems, parallel programming and applications. Clusters come in several major flavours depending on their purpose [14]: High Performance Computing (HPC), High Throughput Computing (HTC), High

Availability (HA) and High Performance Service (HPS). Most cluster computing courses are oriented towards HPC and HTC, usually ignoring HA and HPS. However, most cluster installations, especially Internet data centers, fit into these latter categories. In [22], we described the contents of a master-level cluster computing course that already addresses HPS/HA issues. In this paper, being aware of the growing importance of these kind of servers, we propose an updated version of the course with additional material on HPS/HA at the expense of sacrificing parallel programming topics, that can be taught in other subjects on programming. We describe in detail those topics related to HA and HPS clusters that should be included in the syllabus. A distinguishing feature of the proposed course is that it includes several hands-on sessions to easily assimilate the learned concepts. In particular, we propose building a computer cluster in the lab, installing the operating system, services and required packages. The cluster may be configured taking into account one or more of the aforementioned flavours. In this paper, we propose configuring and deploying a small cluster-based Internet server that provides web service.

As stated in [22], the use of virtualization environments allow students to build their own cluster on the computers of the lab or even on their own laptops. The computing power and memory available on current off-the-shelf computers are high enough to deploy a cluster composed of several virtual machines. Machine management (boot, reset, power-off or replication) is easy to achieve, without interfering with the work of other students. Root privileges are easily fulfilled.

The main contributions of this paper are i) the proposal of selected top-

ics on High Availability and High Performance Service to be included in a cluster configuration and administration course; ii) considering a practical and hands-on approach where students build, configure, test and evaluate a cluster-based web server; iii) using packages and tools used in real installations; and iv) that it tries to fill the gap observed in many cluster computing courses.

The rest of the paper is organized as follows. Section 2 briefly describes the syllabus of the subject. Section 3 includes well-known and established concepts related to high performance service and high availability that should be taught in the subject. Some of these concepts are applied in Section 4, describing the steps followed to build, configure, test and evaluate a small cluster proving web service. Section 5 discusses some issues on teaching the subject and Section 6 gives pointers to some related work. Some conclusions finish the paper.

2. A subject on computer clusters

As cluster computing is an advanced subject [16], the proposed course is aimed to graduate or advanced undergraduate students in Computer Science or Computer Engineering. Expected student previous knowledge includes topics usually covered in these studies as programming, computer organization and architecture, computer networks and operating systems.

We have taught a course with quite similar content for several years in a Master postgraduate course [23], lectured by the Computer Engineering Department at the Universitat Politècnica de València. Lately, we have updated and extended the content related to HPS, trying to improve the student skills

in this important area, with the aim of better preparing them to enter the labour market.

2.1. Syllabus

This section summarizes the lecture topics. A more detailed explanation can be found in [22]. Comparing the subject proposed in this paper with that course, the content about parallel programming has been eliminated, since it is already developed in other courses. This allows us to extend the time devoted to High Performance Service, including new topics. The proposed subject is composed of the following lessons:

1. **Introduction to clusters**. What they are, why they are useful and popular. Cluster classification.
2. **Selection of cluster components**. Main components of nodes, interconnection network, storage system, auxiliary components.
3. **System infrastructure**. Fast review of useful network services for clusters (DHCP, NTP, DNS, NIS, SSH, NAT). Efficient IP address assignment. Improving network performance (channel bonding and jumbo frames).
4. **System installation**. Criteria to select the most appropriate Linux distribution. Boot process and boot loaders. PXE.
5. **Storage systems**. Alternatives for disk array management (LVM, RAIDs, DRBD). Protocols to connect computers and storage, directly (SCSI, SAS) or through a network (Fibre Channel, ISCSI, FCoE, InfiniBand).

Storage architectures: Network Attached Storage (NAS) and Storage Area Networks (SANs). NFS protocol. Cluster file systems (OCFS2, PVFS2 and GlusterFS).

6. **Running applications on a cluster**. Batch execution systems (HTCondor, Slurm) and support for running parallel applications (OpenMPI, OpenMP).
7. **Internet server cluster (I): Load balancing**. Load balancing basics (topologies, operation at levels 4 and 7). Case studies: IPVS and HAProxy. Packet routing. Load balancing algorithms. Sticky sessions.
8. **Internet server cluster (II): High Availability (HA)**. How to get HA in a cluster. HA with HAProxy. Keepalived and corosync. Problems in case of failure: split-brain, fencing and stonith. Cluster Resource Managers: pacemaker.

In this paper we will focus on the content related to the last two topics. The course aims at a deep learning approach as it is considered in [10], where students are interested and try to understand what they are studying. In order to achieve this goal, main concepts explained in theory sessions are applied to a case study that students develop as a lab project. This hands-on part will be described in section 4.

3. Alternatives to configure a High-Performance Service cluster

This section describes the main contents about load balancing and high availability explained in lessons 7 and 8 of the course.

Access to clusters is usually done through a director or load balancer (LB), which is responsible for distributing the client requests among a set of

servers that are actually implementing the offered services. These nodes are usually referred to as the real servers. It is also critical to assure the proper operation of the LB, which otherwise would become a single point of failure in the system. For this reason, the LB is usually replicated.

The LB can be implemented as a specific purpose hardware device or by software on a general purpose computer. While hardware devices offer high performance, they are often expensive and, above all, closed solutions. Alternatively, good open software solutions exist in the market to perform load balancing and high availability. This section describes some relevant possibilities.

3.1. Load Balancing

Load balancers can work at layer 4 or layer 7 of the OSI Model. Load balancing at layer 4 uses only information about IP addresses and transport ports to make its decisions. Working at layer 7 allows the LB to use, additionally, information of the application layer, enabling more specific decisions. To this end, application proxies are usually used. Besides the OSI layer, two more parameters allow to characterise the load balancers. First, the minimum number of network interfaces required by the LB and, second, how the packets are redirected between the LB and the real servers. Both parameters are related to each other.

Clients usually gain access to the cluster through a Virtual IP address (VIP), associated to the LB network interface. This VIP is set in addition to the real IP address bound to the interface. The VIP can float between both LBs when high availability is used. When the LB requires two network interfaces, a pair of VIPs, external and internal, must be used.

At this time, the two most important software projects for load balancing are Linux Virtual Server Project (LVS), and in particular, its module IP Virtual Server (IPVS) [17] and HAProxy [11]. Both of them are fast, reliable and very scalable solutions that can accept thousands of connections simultaneously.

3.1.1. IPVS

IPVS has been part of the Linux kernel for several years. It provides a level 4 solution to load balancing, simple and efficient, for services based either on TCP or UDP.

IPVS allows us to redirect the packets towards the real servers using three different methods: NAT, Direct Routing (DR) and across IP encapsulated tunnels. All of them have an important advantage, they allow the real servers to see the IP address of the client.

NAT is the simplest way to configure IPVS and permits the real servers to use private IP addressing. With this configuration, both incoming and returning traffic must traverse the LB in order to translate the IP addresses (from the external VIP to the real server address and vice versa). For this reason, it is the routing method that puts the greatest pressure on the LB. Moreover, it requires that the real servers have configured the LB as their network router. However, it is one of the most used due to its simplicity. For this reason, we have used it in the lab project. NAT mode is usually deployed connecting the LB to two different IP subnets.

With the second option, Direct Routing (DR), only incoming traffic has to traverse the LB. Afterwards, the real servers can answer directly to the clients. This behaviour can improve performance but requires that not only

the LBs but also the real servers have the VIP as an alias. When an incoming packet reaches the LB, it changes the destination MAC address of the packet by the selected real server MAC. Then, the packet is sent. Thus, the LB has only to modify link level information, which is simple and fast. Moreover, in this configuration, the LB and the real servers are in the same subnet and need only one network interface. However, the main disadvantage is that the real servers should not answer the ARP requests for the VIP. Otherwise, incoming packets may not traverse through the LB. The solution to the ARP problem [27], [2] depends on the operating system and may not be friendly. For the sake of simplicity and time, we will neither test this nor the following configuration in the lab project.

Finally, the IP tunnels are usually the least used strategy. In this case, the LB encapsulates the incoming packet adding the destination IP of a real server. Therefore, the real servers may be in a different LAN than the LB. The answer is sent directly to the client as in DR.

3.1.2. HAProxy

HAProxy restricts its scope to TCP-based services. It has a great reputation [12] as a fast, efficient and flexible solution. For this reason, HAProxy is used in many professional environments with a large number of customers. In addition, it provides high availability services for the real servers.

Usually, HAProxy is configured to work at level 7 but can also operate at level 4. In either of the two levels, as it is a reverse proxy, the client access to the offered services requires two TCP connections. One from the client to the LB and another between the LB and the real server. That is why the real servers can not see the source IP address of the client. If the real servers need

to know this address, there are several solutions. Only for HTTP traffic, an *X-Forwarded-For* header can be added by the LB. Other possibilities are the use of the transparent proxy mode or the PROXY protocol [26].

However, proxies have advantages regarding network configuration. Since the TCP connection is always established between the LB and the real server, the LB will always receive the answer of the real server. In this way, it is not necessary to configure the LB as the router in the real servers. In fact, the LB may even be on a different subnet than the real servers and may have one or two network interfaces. In the lab project, we will use a configuration with two network interfaces, since for the sake of simplicity, the same LB will host both IPVS, working in NAT mode, and HAProxy.

3.1.3. Load Balancing Algorithms

Both IPVS and HAProxy support several load balancing algorithms. Among them, we can mention round-robin, least-connection and source or destination hash. Both of the first algorithms have weighted versions that distribute the requests proportionally to each real server weight. In this way, real servers with different hardware configurations can be used.

All the enumerated algorithms and their advantages are briefly explained in the lectures. For example, source hash is usually used when affinity is desired while destination hash is useful in proxy-cache server clusters.

3.1.4. Persistence and Affinity

Some application protocols work being spread over several TCP connections. HTTP is a well known example. In some cases, for the session to work properly, the server must be aware of the information about all the TCP con-

nections. For example, in sessions where the user has to be authenticated. This requirement can be a problem when a load balancer is used but can be avoided using different strategies.

With session replication, the session information is replicated to other cluster nodes immediately. Another possibility is central storage. In this case, session data are stored in shared storage where all the cluster nodes have access. These two methods are the most robust against a possible failure of the real server that is in charge of the session.

Finally, server affinity or persistence are the most general solutions. In this case, the LB sends all the connections that come from a given user to the same real server during a period of time (sticky sessions). The difference between affinity and persistence lies in the kind of information used. Server affinity is supported by network and transport layer information. For example, the IP address of the client and server port. It is simple but it is not the most advisable way to get stickiness. Persistence uses information of the application layer. Therefore, it can only be used with LBs working at level 7. With HTTP, persistence that takes advantage of cookies is the most frequently used. Its usage will be shown in the lab project.

3.1.5. High Availability

Broadly speaking, adding more real servers to the cluster will improve the load that the system can accept and will increase its reliability. However, in order to work properly, the load balancer needs to know the status of the real servers. In this way, the LB can eliminate from the load distribution those ones that are not longer available. The tools to know the health of the servers can be part of the load-balancing software, as in HAProxy, making

administration easier. In other cases, they have to be added as additional packages as it happens when IPVS is used.

Of course, given its central role, it is also advisable at least to duplicate the LB so that it does not become a single point of failure. This is usually done using an active-passive configuration of LBs and monitoring the status of the active LB. Moreover, the active LB needs to have an additional IP address (i.e., the VIP address) for each interface (internal and external). Should a failure occur, both the VIP and the load balancing service will migrate to the spare LB (or one of them in case there are several available). The new owner of the service will bind its MAC to the VIP in the cluster through an ARP announce.

Both HAProxy as well as IPVS require additional software to check the health of the active LB. To this end, a simple and widely used possibility in Linux systems is the keepalived facility. Keepalived [20] implements the *Virtual Redundancy Routing Protocol* (VRRP) [33] in order to perform load balancer failover.

In addition, keepalived may be used to monitor the status of the real servers and modify the IPVS configuration to be consistent with the health of such servers. Like HAProxy, keepalived incorporates different kind of controls to check the servers.

Both characteristics of keepalived, VRRP and real server health monitoring, can be used together or separately. In the lab project, when HAProxy is used to handle a service, it will distribute requests to the real servers as well as it will monitor their health. However, load balancer failover will be performed using the implementation of VRRP offered by keepalived. In the case

of services managed by IPVS, both tasks will be controlled by keepalived.

In addition to the previous mechanisms, if you want to use more powerful techniques like fencing and stonith, you will need some hardware (Power Distribution Units, PDUs) and specific high-availability software, more powerful but more complex to configure. Typically, this software is known as a cluster stack. It is structured in two layers, located between the application and the transport layer: Cluster Communication and Cluster Resource Management.

At this point, in the lectures, characteristics and tasks performed by both layers, Cluster Communication and Cluster Resource Manager, are detailed. The concepts of quorum, split brain and fencing are also explained. Examples in the Linux world are shown. As cluster communication software, corosync [9] is mentioned, whereas as a cluster resource manager, pacemaker [24] scope is presented.

4. Lab Project

In this section, we describe a case study of a small cluster providing High Performance Service. In particular, a web server will be deployed. As stated above, this academic project will be based on the use of virtual machines. The students build from scratch a cluster of virtual machines, installing and configuring all the required services.

Care must be taken to reduce the size of the cluster as well as the specifications of each node to be representative of actual requirements but also keeping complexity under reasonable limits. The virtual nodes of the cluster will be run either on the desktop computers of the lab or on the students' own laptops. At the time of writing, machines with a 4-core processor, 4 or 8

GB of RAM and more than 500 GB of disk storage are common. We propose a cluster composed of 6 nodes: two LBs, three real servers and one storage node. As stated above, the two LBs provide high availability. The real servers specifically provide the required service. The number of real servers could be increased if more resources are available on the host machine. Finally, the storage node provides a shared storage for user and system data.

Concerning the specifications of each node, it depends on the requirements of the operating system and applications. As the cluster is only used for academic purposes, we only consider the minimal requirements of the operating system. At the time of writing, for Ubuntu 16.04 Server LTS, a minimum 128 MB (512 MB recommended) of RAM and 2 GB of storage are required for a system without desktop. We have successfully worked with uniprocessor configurations with 512 MB of RAM at the LBs and 256 MB of RAM at the real server and storage nodes. An 8 GB virtual disk is used in all nodes but the storage one, where 10 GB are provided.

The lab project consists of several steps. First, the virtual machines that will become the nodes of the cluster will be created and configured. Next, the operating system and basic services will be installed on the cluster. Once the cluster is configured and working, we install and configure the corresponding software tools to provide load-balancing and high-availability that allow the cluster to work as a web server. This paper only deals with this latter part of the project. Please see details for operating system installation and system configuration in [22].

Regarding load balancing, we will consider both HAProxy and IPVS based approaches. As stated in Section 3, we will consider only NAT mode

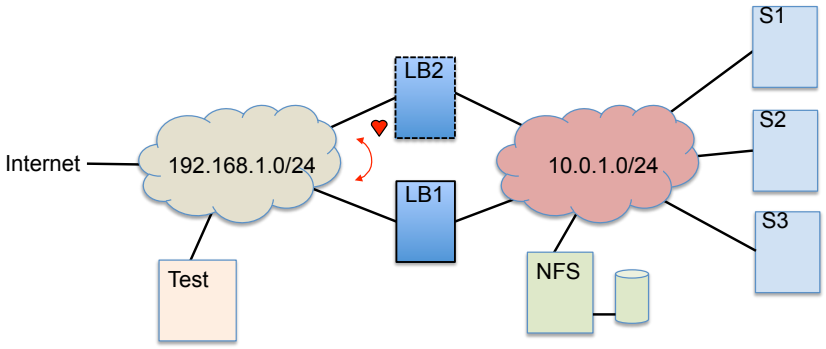

Figure 1: Block diagram of the cluster

in the lab project. It must be noticed, though, that advanced students could extend the project with other configurations.

Figure 1 shows the block diagram of the Internet server.

As it can be seen, the cluster-based server consists of two load balancers (LB1 and LB2), three real servers (S1, S2, S3) and one storage node (NFS). All the nodes are attached to the same "internal" subnet. Storage node provides shared access to all cluster nodes. Notice that LB1 and LB2 have two network interfaces to allow communication between the cluster and the external world. In an actual cluster site, the "external" subnet could be located behind the firewall. Nodes LB1 and LB2 are configured as active–passive, sharing the VIPs that will be up only on the active node. To test different alternatives, we will configure three external VIPs to the system:

- VIP-H1: 192.168.1.200. HAProxy-based load balancer working in http mode.
- VIP-H2: 192.168.1.201. HAProxy-based load balancer working in tcp mode.

Node	IP Address	
LB1	192.168.1.110 10.0.1.110	VIP-H1: 192.168.1.200 VIP-H2: 192.168.1.201 VIP-IPVS: 192.168.1.202 VIP-INT: 10.0.1.200
LB2	192.168.1.111 10.0.1.111	VIP-H1: 192.168.1.200 VIP-H2: 192.168.1.201 VIP-IPVS: 192.168.1.202 VIP-INT: 10.0.1.200
S1	10.0.1.10	
S2	10.0.1.20	
S3	10.0.1.30	
NFS	10.0.1.100	

Table 1: IP addresses of the cluster nodes.

- VIP-IPVS: 192.168.1.202. IPVS-based load balancer.

In addition, as real server answers must return through the LB, another internal VIP address (VIP-INT) is also required.

To better differentiate both subnets, we have chosen address block 192.168.1.0/24 for the external network and 10.0.1.0/24 for the internal network. Table 1 shows the assigned IP addresses.

4.1. HAProxy installation and configuration

Both LBs must run HAProxy [11]. HAProxy is a very powerful load-balancing package, which can be configured with a lot of options. In the lab project, we configure basic services with demonstrative purposes. Figure 2 shows the proposed configuration in the lab project.

The configuration file is composed of several sections. The most important ones are detailed below:

frontend These sections define the services that will be handled by HAProxy. Each frontend section defines a service that will be handled by a set of servers identified in the "backend" section indicated here. For each defined service, the socket address, the load balancing mode (http, layer 7 or tcp, layer 4) and the associated backend must be specified. In this case, we define two services:

- haproxy-http with socket address 192.168.1.200:80, which distributes requests in "http" mode. As the LB understands the application protocol, requests can be modified. The "forwardfor" option instructs HAProxy to store the client IP address in a field of the http request that is forwarded to the real servers.
- haproxy-tcp with socket address 192.168.1.201:80, which distributes requests in "tcp" mode.

backend These sections define sets of servers that will be in charge of the service offered in frontend sections. In this case, we define two server sets ("http-servers" and "tcp-servers") that are composed of all available real servers. The first server set cyclically distributes requests among the real servers. In this case, to deal with session persistence, cookies are used. The second server set uses the client IP address to enforce server affinity.

listen This section allows grouping frontend and backend definitions. In this case, it is used to configure the HAProxy stats page.

4.2. Keepalived installation and configuration

Both LBs must run keepalived service [20]. As we have stated before, in the lab project, we configure basic services with demonstrative purposes. Figure 3 shows the proposed configuration in the lab project. It corresponds to LB1 node.

The configuration file is composed of several sections:

vrrp_instance This section configures a virtual interface. Virtual interfaces will allow LB failover using VRRP protocol. One or several VIP addresses are assigned to the virtual interface. In this case, we define the two virtual interfaces, "VI_E" and "VI_I", corresponding to the external and internal networks, respectively. The associated physical network interface is also specified. The configuration shown corresponds to LB1, which will be initially in charge of services (state MASTER). Node LB2 is configured with state BACKUP, and will activate VIP addresses only if LB1 fails. In case both nodes are active, the one with more "priority" will be promoted to MASTER.

vrrp_sync_group In case of MASTER LB failure, all the configured VIP addresses should failover to the BACKUP LB node. In our configuration, this includes both external and internal VIP addresses. This section groups virtual interfaces whose VIP addresses will be jointly migrated in case of failover.

virtual_server Keepalived package also includes a wrapper to configure load-balancing using IPVS. Services are configured in this section, specifying the socket of the service (192.168.1.202:80 in our case), the rout-

ing method (NAT), the load balancing algorithm (round-robin) and the pool of real servers.

4.3. Additional Configuration

Having an operational cluster involves manipulating different services and their associated configuration files. For the sake of brevity, we will only describe those ones that either can be tricky or specially oriented to the HPS cluster. Of course, network interfaces of all cluster nodes should be configured, and services like DNS or NFS should also be working.

HAProxy process will be started on both LBs at boot time. However, only one (the MASTER) will have the VIP addresses configured. This may generate a problem in the BACKUP LB node, as HAProxy wants to bind to a non-existent IP address. This is easily solved by setting the "net.ipv4.ip_nonlocal_bind" kernel variable, which allows a running load balancer instance to bind to an IP that is not local for failover.

Although HAProxy does not require NAT service (in fact, the load balancer is a proxy), the chosen configuration of IPVS does. Therefore, both LB1 and LB2 should allow NAT. It is configured by setting the "net.ipv4.ip_forward" kernel variable and correctly configuring the nat table with iptables command.

Real servers must have configured the provided service (web in our lab project). We use the well known apache package, which is easy to configure. To access the "X-Forwarded-For" header, the apache remoteip module must be enabled.

4.4. Testing the system

To test the HPS cluster, we will issue some requests to the system. In an actual system, the requests would be issued from any computer to the public IP address of the server. In the lab project, though, we have a cluster of virtual machines that is behind the virtualization environment. Although this environment could be configured to forward requests to the virtual machines, we propose to add a new node to the cluster: the Test node. This new machine will be attached to the external network (please see Figure 1). Therefore, we will issue requests to the HPS cluster from the Test node. Among the advantages of this approach, we are free of installing and configuring whatever benchmark programs we want, and the fact of being so close to the HPS cluster allows the possibility of performing its evaluation under heavy traffic.

In order to verify the correct system behaviour, the following tests should be performed:

- The web server works without any component failure. The active LB distributes the requests among the real servers.
- The web server works when one or several real servers fail. The survivor real servers will be in charge of the service.
- The web server works when the active LB node fails. The backup LB node takes charge of the load-balancing task.

4.4.1. System without failure

As the first test, we will check that only one LB has the VIP addresses configured. The ip command (ip addr) does the work. When the system has

just booted up, only LB1 will have the VIP addresses configured.

Next, we will perform some tests to check load balancing. We propose to write a short PHP program (see Figure 4) that shows the current date and time, the server IP address, the client IP address and the X-Forwarded-For header. All this information is available in the PHP $_SERVER variable. In addition, the set of cookies (PHP $_COOKIE variable) can be also displayed. Once the program is installed on the system, we will issue some requests with the help of a browser. We use the w3m text-based browser.

The results should be the following:

- 192.168.1.200. HAProxy, http mode. Although the load-balancing algorithm is set to round-robin, as cookie-based persistence has been set, all the requests are assigned to the same real server. The assigned server is shown in the SERVERID cookie. The active LB is the client seen by the real servers, and the true client IP address is shown in the X-Forwarded-For header, as HAProxy is working in http mode and the forwardfor option is set.

- 192.168.1.201. HAProxy, tcp mode. As the load balancing algorithm is set to "source", all the requests are also assigned to the same real server, chosen as a function of the client IP address. The active LB is the client seen by the real servers, and the true client IP address is unknown as HAProxy is working in tcp mode.

- 192.168.1.202. IPVS. As the load balancing algorithm is set to round-robin, each request is handled by a different real server. The true client IP address is correctly seen by the servers.

As an alternative, the HAProxy status page can be also displayed to check that all real servers are working and the applied load balancing algorithm. For IPVS-based service, the ipvsadm utility shows its status. Finally, students are also encouraged to explore log files as they are very instructive.

4.4.2. System with real server failure

A real server failure can be easily forced by (virtually) disconnecting the network interface. After the failure, the server node will not longer receive any request. New requests will be assigned to any of the survivor real servers. HAProxy stat page will clearly show that the real server is "DOWN". On the other hand, the ipvsadm utility will not longer show the failed real server.

Service will continue, provided that there is at least one real server available. A fallback real server could also be configured.

4.4.3. System with LB failure

Likewise real servers, failure of the LB could be easily forced by disconnecting one of its network interfaces. As both interfaces are grouped in the keepalived configuration, failover of both external and internal VIP addresses will occur. Once the fault is forced, it is easy to check VIP address migration by issuing the ip command in both LB nodes. The formerly BACKUP LB node will now assume the MASTER role. Log files will also store the failover.

If we issue again some requests to the VIP addresses, everything should be fine. The operating LB is now in charge of the load-balancing.

4.5. Evaluation of the web server

The HPS cluster is built with virtual machines. Therefore, it is not expected to achieve high performance results as in a real system. However,

it is a goal of the subject to teach the students how to evaluate such a system. Usually, HPS servers are characterized in terms of their throughput, measured in processed requests per second. Response time of each request is also useful.

As in other cases, performance is ultimately limited by the bottleneck of the system. In an Internet server, the bottleneck could be located at the real servers, the storage, the LB or the network. System optimization alternatives depend on the bottleneck location. In the lab project, we work in an scenario where the bottleneck is located at the real servers. This scenario is compatible with the implementation of the cluster as virtual machines, and, as we will see, it justifies the use of a cluster to improve performance.

4.5.1. Evaluation model

Benchmarks are often used to evaluate system performance. In the web server evaluation arena, there are several options, like apache benchmark [1], siege [29] or JMeter [18]. In the lab project, we propose using apache benchmark because it is easy to use.

Apache benchmark is a tool for benchmarking an HTTP server. The benchmark basically issues a given number of concurrent requests to the server, waiting for the response. This cycle is repeated until a given number of requests have been processed or an amount of time has elapsed. The faster the server, the higher the number of requests per second obtained.

The benchmark is launched to a test page. In an actual system evaluation, a set of tests containing different scenarios should be evaluated. In general, a data access application where storage and database accesses and also some computing are required could be used. However, to keep the anal-

ysis simple, as stated above, in the lab project, we use a test page that puts the pressure on the real servers. In particular, we prepared a PHP test page that performs some compute-intensive task. We have used with success the code shown in Figure 5, which computes the π constant by numerical integration ($\pi = \int_0^1 \frac{4}{1+x^2} dx$) using the midpoint rectangle rule. The client requests pass the number of considered rectangles to the URL, therefore adjusting the computational complexity of the page.

4.5.2. Evaluation results

As a first step, we will analyse the execution time of the π constant compute program using numerical integration as a function of the number of rectangles. We will select a number of rectangles high enough to put real servers under pressure. In our experiments, 10^6 rectangles required roughly 0.6 secs. This means that one real server (with one core) will serve only $\frac{1}{0.6} = 1.67$ requests per second.

Then, apache benchmark is used to test the system. For instance, 100 requests with concurrency levels of 1, 2 and 3 are issued. Remember that our cluster has 3 real servers. Another test where the number of concurrent requests is twice the number of CPUs (6 in our case) is also issued. For comparison purposes, the performance of only one node is also evaluated. To do so, all the real servers but one are shut down and the same tests are performed. Figures 6 and 7 show the obtained results.

As we can see, the cluster is able to scale up throughput with the number of concurrent requests, up to the number of real servers. From this point onwards, throughput does not grow any more. Regarding response time, it is kept constant with the increase of the number of concurrent requests, up

to the number of real servers. If more concurrent requests are present, some nodes will be assigned more than one request. In particular, for twice the number of real servers, each real server has to deal with two requests, and response time grows to twice the initial one.

The results obtained with only one node confirm this behaviour. As soon as there are several concurrent requests, the real server processing power is shared among them, thus linearly increasing response time. Throughput does not grow, as the real server is saturated with just one request.

This experiment clearly justifies how computer clusters allow server scaling for high performance service. Remember, though, that the pressure is put on the real servers, as handling the requests requires some processing time.

The experiment could be repeated by requesting the same page but with a lower number of integration rectangles. The lower this number, the lower the processing requirements at the real servers. As a consequence, response time will be reduced and throughput will be accordingly increased. As the number of requests per second increases, the pressure on the load balancer also increases. In fact, the load balancer may become the bottleneck of the system. Load balancer performance can be improved in several ways, but it is out of the scope of the subject. Interested students may analyze in depth this issue in their master thesis.

5. Teaching Discussion and Assessment

The proposed lab project is challenging for students as they have to tackle a quite real problem: installation, configuration and evaluation of their own

computer cluster. Although we provide the students with a complete documentation to achieve this goal, problems usually arise. Students are encouraged to try solving the problems by themselves, carefully analyzing system error or warning messages, log files, and also browsing specialized Internet sites ([30], [21], [28], [11]) to look for solutions. This latter source of information is very valuable, as when looking for some particular question, some related topics that may draw student attention are discovered. On the other hand, despite the fact that each student works in his/her cluster, they usually help each other, which allows teamwork among them.

The authors have been teaching a similar subject [23] for several years in a Master postgraduate course. The subject is targeted to bachelors in Computer Science, Computer Engineering or Telecommunication Engineering. It has 40 assigned hours in the syllabus, spread over 16 sessions of 2.5 hours each. We reserve six complete sessions throughout the course to install and configure the computer cluster using virtual machines. A total of 15 classroom hours is devoted to deploy the computer cluster by the students. As stated above, the contents developed in this paper extend the HPS part of the subject. In this way, we are adapting its contents to fill the detected gap regarding High Performance Service in high performance computing education curricula, trying to adapt the contents taught to actual market reality.

Student feedback over years has been really good, as students love solving practical problems. They are very motivated in the hands-on project. As the project starts from scratch and ends with a completely operating cluster offering High Performance Service, they are very proud of their work as they feel that they had a complete control of all the involved steps, overcoming

all the arisen problems. In particular, students are very excited when they understand the importance of load balancing on server throughput and response time, analysing how HTTP requests are being balanced or a failover is produced in the LB. Overall, student opinion is that they have learned practical and useful issues of today servers.

We have evaluated the course considering two different aspects: contents and methodology. Regarding to the former, students are asked to complete a survey before starting the course in order to know their previous knowledge about the subject. Then, at the end of the course, they have to take a written exam (30%). The topics of the exam include all the contents taught during the course, making special emphasis on the lab project aspects. In addition, the deployment of the computer cluster over the virtual machines is also considered in the assessment (35%). Table 5 shows the initial survey results for the questions related to load balancing and high availability. In this table, 1 (-) value means "I don't know anything about it" and 5 (+) means "I absolutely know them in depth". As we can see, most of the students know nothing or have little knowledge about these concepts. None of them have ever installed or configured this kind of tools. However, all them had their cluster working correctly at the end of the term. Moreover, regarding the final exam, the results of the questions related to these topics were very good. More than 80% of the students got marks of at least 70% of the question values (equivalent to level B mark). As an example, 55% of the students answered perfectly the questions about fencing. As table 5 shows, none of them knew nothing about it at the beginning of the course. Other questions about high availability and load balancing (topics A and C in the

previous survey) got 82% answers with the maximum score.

Concerning the methodology assessment, the results obtained in the yearly survey of the University confirm student satisfaction. A five-point Likert scale was used in the questionnaire, with the typical format (Strongly Agree, Agree, Undecided, Disagree, Strongly Disagree). Table 5 shows the results in some of the questions related to methodology and teaching materials. As we can see, most of the results had highly positive scores with percentages approaching 80% of Strongly Agree.

Statement	1 (-) %	2 %	3 %	4 %	5 (+) %
A. I can explain in depth the tasks performed by a load balancer	37	63	0	0	0
B. I know some load balancing software tool	87	13	0	0	0
C. I can explain in depth the concept of high-availability	25	75	0	0	0
D. I can explain in depth the purpose of fencing techniques	100	0	0	0	0

Table 2: Survey conducted to the students about previous knowledge.

Statement	Strongly disagree %	Disagree %	Undecided %	Agree %	Strongly agree %
A. Teaching methodology and classroom activities help the learning process of the student	0	0	0	30	70
B. There is a good working environment in the classroom	0	0	0	20	80
C. The contents of the course seem interesting to the students	0	0	11.11	11.11	77.78
D. Teaching materials are very helpful for learning and reaching the subject goals	0	0	0	22.22	77.78

Table 3: Survey conducted to the students about methodology satisfaction.

6. Related Work

Although several papers have already addressed how to teach cluster computing in university courses, to the best of our knowledge, none of them is focused on load balancing or high availability. However, since the beginning, we find authors like Baker and others [5] underlining its importance in a cluster system. It is also true that there is a wide set of topics that may be covered in a cluster computing course [3]. For this reason, the topics finally chosen will depend on the course objectives.

In [3], several authors present a selection of possible topics for cluster computing courses, based on their experience teaching this subject in different universities in USA and Australia. The proposed material is pretty wide and covers many different aspects of cluster computing. This allows other instructors to select the contents best suited to their course objectives. The paper underlines the importance of students can see how computers in a cluster work together to provide a service. Furthermore, among the typical topics that can be covered in a course, systems like IPVS are included. However, in the actual courses shown as examples in the paper, we find that frequently an important part of the course is focused on parallel programming.

Later works proposed to build clusters to use them in different courses. In [4], [8], [13] and [19], those clusters are implemented using real hardware while [31] and [34] choose virtualization. In general, most courses build the clusters pursuing two goals. First, it allows the students learning how to make and test an HPC cluster [8], [19]. Moreover, in second place, many courses take advantage of the cluster built to run parallel programs [4], [13], [31], [34].

Two of the courses are not addressed to students in computer science or computer engineering. [8] is intended for students in engineering or sciences while the course described in [19] is aimed at business school students. That is why, neither of them include parallel programming. They are more interested in how to build a cluster and in its performance evaluation, since their students may have to develop clusters in their workplaces. In [8] cluster evaluation is performed by running MPI test programs. [19] focuses on the best resource allocation, for example, reallocating RAM between nodes and evaluating cluster performance with each configuration. Additionally, this course includes also some useful contents about systems security.

All the clusters developed in these courses are based in Linux operating system apart from [34]. Overall, they use quite basic configurations that do not represent real life installations. Regarding to the cluster usage, they are intended for high performance computing usually ignoring the high performance service. However, this latter usage represents a very high percentage of cluster installations [14]. Internet servers that handle a high demand on a given service (web, mail, file transfer, database, videos, etc.) are everywhere and are usually based on clusters. Finally, almost no course addresses high availability or advanced cluster storage.

On the contrary, the course proposed in this paper considers both high performance/high throughput computing and high availability/high performance service flavours of computer cluster usage. In addition, load balancing and high-availability are widely developed during the lectures as well as, taking a hands-on approach, in the lab project.

7. Conclusions

Nowadays, the use of clusters has widely spread both among companies and in the academia. Most university cluster computing courses are focused on High Performance Computing, the most common type of clusters in universities. However, big servers on the Internet usually rely on High Performance Service-High Availability (HPS-HA) clusters.

In this paper, we propose a master course on cluster configuration and administration, that tries to fill this gap. It is the updated version of the previous course presented in [22] that has been taught for several years. In this edition, the topics about HPS-HA have been updated and extended. To this end, we selected some of the most interesting alternatives, that are presented in the lectures. In this paper, they have been summarized in Section 3. Moreover, these contents are reinforced through the lab sessions. There, students, starting from scratch, install, configure and test an HPS-HA cluster providing a web service. In order to train the students for their working life, the whole process uses packages and tools widely used in companies. In addition, the lab project allows students to reinforce several important issues. First, how to check if the cluster will work properly after a failure. Next, how clusters allow to scale up the server load. And finally, how the performance bottleneck can be in different points of the cluster.

Students remain highly motivated during all the project but, specially, at the end, when HPS and HA tests show cluster performance improvement and resilience, respectively.

Acknowledgements

This work was supported in part by the Spanish Ministerio de Economía y Competitividad (MINECO) and by FEDER funds under Grant TIN2015-66972-C5-1-R.

[1] ab - Apache HTTP server benchmarking tool, https://httpd.apache.org/docs/2.4/programs/ab.html (Accessed 10-06-2017).

[2] ARP problem in VS/TUN and VS/DR, http://www.linuxvirtualserver.org/docs/arp.html (Accessed 25-07-2017).

[3] A. Apon, R Buyya, H Jin, J Mache. Cluster computing in the classroom: topics, guidelines, and experiences. In: IEEE/ACM Int. Symposium on Cluster Computing and the Grid, pp. 476-483, (2001).

[4] S. Aydin and O.F. Bay. Building a high performance computing cluster to use in computing course applications. *Procedia - Social and Behavioral Sciences*, 1(1): pp. 2396-2401, (2009).

[5] M. Baker, A. Apon, R. Buyya and H. Jin. Cluster computing and applications. *Encyclopedia of Computer Science and Technology*. New York: Marcel Dekker, Aug. 2001, vol. 45.

[6] D.J. Becker, J. Salmon, T. Sterling and D.F. Savarese, *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters*, (MIT Press, 1999).

[7] R. Buyya, *High Performance Cluster Computing: Architectures and Systems, Vol. 1*, (Prentice Hall, 1999).

[8] M-H. Chen and T-L. Li. Construction of a High-Performance Computing Cluster: A curriculum for Engineering and Science Students. *Computer Applications in Engineering Education* 19(4), pp. 678-684. (2009).

[9] Corosync. The Corosync Cluster Engine, http://corosync.github.io/corosync/ (Accessed 20-06-2017)

[10] D. H. J. M. Dolmans, S. M. M. Loyens, H. Marcq and D. Gijbels. Deep and surface learning in problem-based learning: a review of the literature. *Advances in Health Sciences Education*, 21(5): 10871112. (2016) DOI: 10.1007/s10459-015-9645-6

[11] HAProxy, http://www.haproxy.org/ (Accessed 30-05-2017)

[12] HAProxy. The Reliable, High Performance TCP/HTTP Load Balancer, https://www.haproxy.org/they-use-it.html ((Accessed 25-07-2017)

[13] V. Holmes and I. Kureshi, Developing High Performance Computing Resources for Teaching Cluster and Grid Computing courses. In: ICCS International Conference On Computational Science (2015). doi: 10.1016/j.procs.2015.05.310

[14] D.C. Hyde, M. Baker, editor. *Cluster Computing White Paper* pp. 110-119, (2000). Available from http://arxiv.org/pdf/cs/0004014.pdf.

[15] Computer Engineering Curricula 2016 (draft). Association for Computing Machinery (ACM)/IEEE Computer Society.

(2015) https://www.computer.org/cms/Computer.org/professional-education/curricula/ComputerEngineeringCurricula2016.pdf

[16] Prasad, S. K., Chtchelkanova, A., Dehne, F., Gouda, M., Gupta, A., Jaja, J., Kant, K., La Salle, A., LeBlanc, R., Lumsdaine, A., Padua, D., Parashar, M., Prasanna, V., Robert, Y., Rosenberg, A., Sahni, S., Shirazi, B., Sussman, A., Weems, C., and Wu, J. 2012. NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing - Core Topics for Undergraduates, Version I, Online: http://www.cs.gsu.edu/ tcpp/curriculum/index.php, 55 pages.

[17] IP Virtual Server https://en.wikipedia.org/wiki/IP_Virtual_Server (Accessed 30-05-2017)

[18] Apache JMeter™, http://jmeter.apache.org/ (Accessed 10-06-2017)

[19] F.L. Kitchens, S.K. Sharma and T. Harris, Integrating IS Curriculum Knowledge through a Cluster-Computing Project - A successful Experiment. *Journal of Information Technology Education*, 3, pp. 263-278, (2004).

[20] Keepalived, http://www.keepalived.org/ (Accessed 07-06-2017)

[21] https://www.loadbalancer.org/resources/deployment-guides (Accessed 26-07-2017)

[22] P. López, E. Baydal, On a course on computer cluster configuration and administration, in: J. Parallel Distrib. Comput. 105 (2017) 127-137. http://dx.doi.org/10.1016/j.jpdc.2017.01.009

[23] Master's Degree in Computer and Network Engineering, https://www.upv.es/titulaciones/MUIC/index-en.html (Accessed 15-05-2017).

[24] Pacemaker, http://clusterlabs.org/ (Accessed 20-06-2017).

[25] G.J. Pfister, *In Search of Clusters*, (Prentice Hall, 1998).

[26] The PROXY protocol. Versions 1 & 2, https://www.haproxy.org/download/1.8/doc/proxy-protocol.txt (Accessed 20-06-2017)

[27] Direct routing, https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Virtual_Server_Administration/s2-lvs-directrouting-VSA.html (Accessed 25-07-2017)

[28] Red Hat: Customer portal, https://access.redhat.com/ (Accessed 25-07-2017)

[29] Siege Home, https://www.joedog.org/siege-home/ (Accessed 10-06-2017).

[30] serverfault, https://serverfault.com/ (Accessed 25-07-2017).

[31] G.K. Thiruvathukal et al. Virtualization for Computational Scientists, *Computing in Science & Engineering*, 12(4), pp. 52-61, (2010).

[32] Top500 Supercomputer Lists, http://www.top500.org/lists (Accessed 12-06-2016)

[33] Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6, https://tools.ietf.org/html/rfc5798 (Accessed 7-06-2016)

[34] R.L. Warrender, J. Tindle and D. Nelson, Development of a Virtual Cluster, In: Int. Conf. on High Performance Computing and Simulation (HPCS), pp. 545-551, (2013).

```
global
    ...

defaults
    ...

frontend haproxy-http
    mode http
    option httplog
    option forwardfor
    bind 192.168.1.200:80
    default_backend http-servers

frontend haproxy-tcp
    mode tcp
    option tcplog
    bind 192.168.1.201:80
    default_backend tcp-servers

backend http-servers
    mode http
    balance roundrobin
    cookie SERVERID insert
    option httpchk
    server www server1:80 cookie S1 check
    server www server2:80 cookie S2 check
    server www server3:80 cookie S3 check

backend tcp-servers
    mode tcp
    balance source
    option httpchk
    server www server1:80 check
    server www server2:80 check
    server www server3:80 check

listen stats
    bind 0.0.0.0:8000
    mode http
    stats show-desc LBx node
    stats refresh 5s
    stats uri /haproxy?stats
    stats realm HAProxy Statistics
    stats auth admin:passwd
    stats admin if TRUE
```

Figure 2: HAProxy configuration file.

```
vrrp_sync_group VG1 {
        group {
                VI_E
                VI_I
              }
        }

vrrp_instance VI_E {
        state MASTER
        interface eth0
        virtual_router_id 51
        priority 150
        authentication {
                auth_type PASS
                auth_pass mypasswd
        }
        virtual_ipaddress {
                192.168.1.200
                192.168.1.201
                192.168.1.202
        }
}

vrrp_instance VI_I {
        state MASTER
        interface eth1
        virtual_router_id 52
        priority 150
        authentication {
                auth_type PASS
                auth_pass mypasswd
        }
        virtual_ipaddress {
                10.0.1.200
        }
}

virtual_server 192.168.1.202 80 {
        delay_loop 3
        lb_algo rr
        lb_kind NAT
        protocol TCP

        real_server 10.0.1.10 80 {
        weight 1
        TCP_CHECK {
                connect_timeout 3
                }
        }
        real_server 10.0.1.20 80 {
        weight 1
        TCP_CHECK {
                connect_timeout 3
                }
        }
        real_server 10.0.1.30 80 {
        weight 1
        TCP_CHECK {
                connect_timeout 3
                }
        }
}
```


Figure 3: Keepalived configuration file for LB1.

```
<?php
header("Refresh: 10");
echo "Date: " . date ("Y/m/d") . "<br>";
echo "Time: " . date ("H:i:s") . "<br>";
echo "Server IP: ". $_SERVER['SERVER_ADDR'] . "<br>";
echo "Client IP: ". $_SERVER['REMOTE_ADDR'] . "<br>";
if ( isset( $_SERVER['HTTP_X_FORWARDED_FOR'] ) ) {
    echo "Forwarded-For: ". $_SERVER['HTTP_X_FORWARDED_FOR'] . "<br><br>";
    } else {
    echo "Forwarded-For: ". "Unknown" . "<br><br>";
    }
echo "Cookies:<br>";
print_r($_COOKIE);
?>
```

Figure 4: PHP code for testing load-balancing

```
<?php
$t1=microtime(true);

$n=$_GET["n"];
$area=0.0;

for ($i=0; $i<$n; $i++)
    {
    $mid=($i+0.5)/$n;
    $area=$area+4.0/(1.0+$mid*$mid);
    }
$result=$area/$n;

$t2=microtime(true);
$exectime=$t2-$t1;

printf ("Computed PI, n = %d, is equal to %f<br>", $n, $result);
printf ("Execution time = %.5f secs<br>", $exectime);
?>
```

Figure 5: PHP code to compute π constant

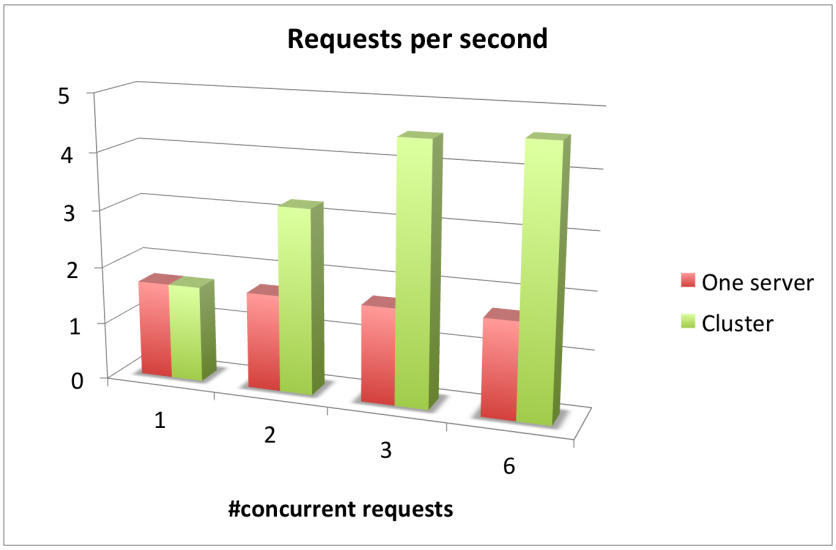

Figure 6: Evaluation of the cluster: requests per second

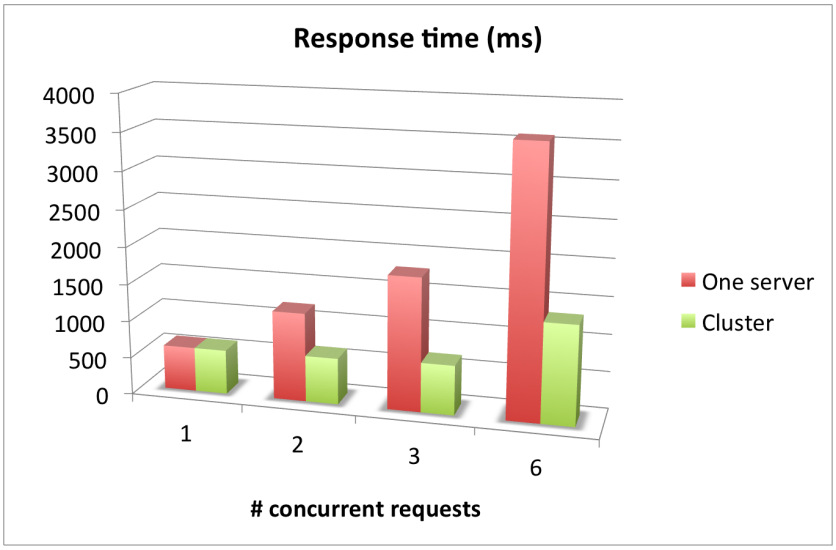

Figure 7: Evaluation of the cluster: response time