Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 679–683

ISBN 978-83-60810-51-4

Building well-balanced CDN[1]

Piotr Stapp, Piotr Zgadzaj
Warsaw University of Technology
Pl. Politechniki 1, 00-661 Warszawa, Poland
Email: p.stapp@mini.pw.edu.pl, p.zgadzaj@mini.pw.edu.pl

***Abstract*—The following document describes building well-balanced CDN evolution process. We start with very intuitive, but unfortunately wrong solution and change it to the one which works almost ideally. We realized our experiments on Planetlab environment, which is a good internet simulation. Every experiment description is in common format for easy comparison. Document include for each experiment methodology: environment description, system architecture, short description of experiments, result analyzing and conclusions.**

I. Introduction

In 2012 Poland and Ukraine hold the UEFA European Cup in soccer. Using historical data, we know that in 1998 official Soccer World Cup Website had 1,35 billion request over 3 months, with peaks 73 million request per day and 12 million request per hour [1] These numbers were exceeded during Summer Olympic Games in 2004 and 2008. One can expect that in 2012 these numbers will be exceeded several times.

Nowadays Content Delivery Network (CDN) is a solution for the above problem. But many servers give us only one thing: possibility of user distribution. The Wikipedia entry for CDN states: "A content delivery network or content distribution network (CDN) is a system of computers networked together across the Internet that cooperate transparently to deliver content to end users, most often for the purpose of improving performance, scalability, and cost efficiency." But an important question is how to improve those? We will try to find a proper answer.

II. CDN Architecture

A. Notation

First of all we need to define a notation. We decided to use the same as in [2], depicted below:

- Web Server (WS) - is a container of content;
- Service Registry (SR) - discovers and stores resources and policy information in a local domain.

B. Architecture (CDN definition)

Using the above notation we can define: „CDN is built with one or more Web Servers (WS'S) and one Service Registry (SR)". In the simplest CDN definition SR works as „first line" for user requests. It can also be responsible for resource discovery and policy in local domains. WS works as a container for content available for user. Architecture of CDN is presented in figure 1 below:

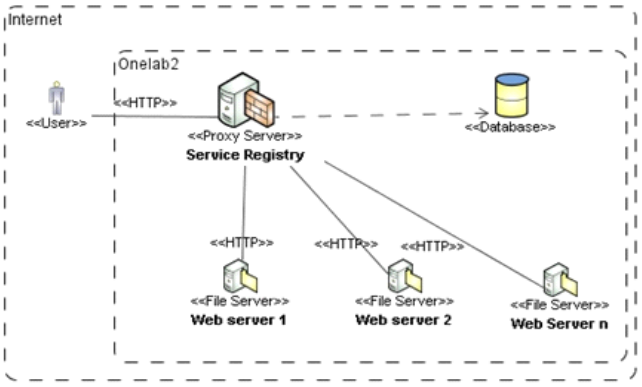

Fig 1. CDN architecture

The main idea is as follows (presented on figure 2 bellow):

- End user sends a request for some content to the Service Registry SR;
- SR finds "the best" Web Server WS for this user;
- SR redirects the user to the "best" WS;
- The WS receives the request;
- The user downloads the requested content from WS.

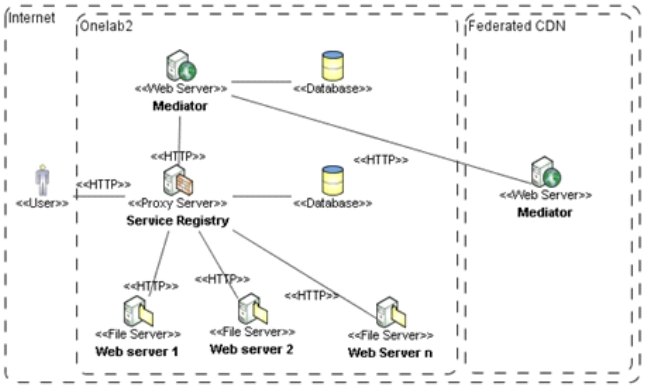

Fig 2. CDN sequence diagram

It is important to notice that WS is described as "the best" depends on the policy in the SR. For example it may depend on GEO-IP combined with WS's load and speed. There is a large number of metrics, but only those based on Quality of Service -related parameters have matured to a level that allows the delivery of comparable results.

C. PlanetLab

The Wikipedia entry for PlanetLab states: "PlanetLab is a group of computers available as a testbed for computer networking and distributed systems research. It was established in 2002 by Prof. Larry L. Peterson, and as of June 2010 was composed of 1090 nodes at 507 sites worldwide. Each research project has a "slice", or virtual machine access to a subset of the nodes."

We can define PlanetLab as an Internet simulation. Unfortunately it has the biggest disadvantage of Internet – it is neither repeatable nor isolated. In other words: every experiment is unique and other experiments performed at the same time have influence on our experiment. To obtain dependable results one must conduct several experiments and observe the average.

III. Environment Set-up

As we have described earlier we created our CDN on PlanetLab network, which uses Linux base OS's. Our environment consists of two main parts, i.e. Service Registry SR and Web Server WS.

1. On Service Registry we have installed some additional software:
   1.1. For handling user http requests and redirections we have used Apache based WWW server (`Lighttpd` [4]), with enabled `FAST-CGI` and `PHP` support (to enable database access) – we installed following packages:
      a) lighttpd;
      b) lighttpd-fastcgi;
      c) php-cli.
   1.2. As a storage for information about network metrics and topology we used SQL database: **PostgreSQL** 8.2.11 [5]. To facilitate operations on storage we created a special api consisting of several SQL-based stored procedures. Database api is described in details in §IV (Internal Architecture)
   1.3. Database was periodically updated by shell scripts configured in **CronTab** [6] which gathers data from Web Servers (WS) about current workload.
2. On Web Server (WS) we have installed the following software:
   2.1. For handling user http requests we have used Apache base WWW server (**Lighttpd** [4]) with enabled FAST-CGI support - hence the following packages were installed:
      a) lighttpd;
      b) lighttpd-fastcgi.
3. We also used some general-purpose tools to facilitate performing tests:
   3.1. For running shell scripts we used PSSH [2]. This tool is similar to standard SSH client, the main difference is that it allows to run shell scripts in parallel (on multiple nodes simultaneously)
   3.2. For transferring binary resources (files) we use PSCP [2]. This tool is an extend of SCP and similar to PSSH, as it allows to transfer one file to multiple nodes simultaneously.
   3.3. For simulating user requests (requests for content) we use WGET [3].

IV. Internal Architecture

In §III (Environment set-up) we have described that CDN workload data is stored in SQL database installed on Service Registry (SR). The database is periodically updated with the data gathered from Web Servers (WS) by a shell script. The shell script loads data into database through special stored procedure. Shell script is scheduled as cron task.

User requests on Service Registry (SR) invoke stored procedure, which extracts from database location of the best Web Server (WS).

The main components of database api are as follows:

- FUNCTION *add_new_weight_value*(character varying, character varying, character varying) – SQL function (used by shell script) which adds new rate value for specified Web Server;
- FUNCTION *recount_agregate_weight*(character varying) – SQL function (used by shell script) which recounts weight of specified Web Server after adding new rate value;
- FUNCTION *get_nearest_cdn*(character varying) RETURNS character varying – SQL function (used by PHP script) which finds "the best" Web Server for the specific users.

V. Experiment 1

A. Environment Modifications

Service Registry

We did not change the architecture on root server. The only modification was shell script which collects data from CoMon service [1]: "CoMon provides a monitoring statistics for PlanetLab at both a node level and a slice level. It can be used to see what is affecting the performance of nodes, and to examine the resource profiles of individual experiments."

In our case CoMon service was the natural way of collecting data.

Database

In this experiment we used database only for storing information about nodes in experiments and results from CoMon. No calculations were performed.

Web Server

Since sending files was done with Lighttpd using PHP and collecting data was done by build-in CoMon service, we did not need any modifications to web servers.

Clients

The client architecture is in accordance with the definition from §VI

Procedure Modification

In this experiment we created a procedure of conducting experiments. It can be described by the following algorithm:

1. Deliver content to every server node
2. Deliver bash script for downloading files to client nodes.
3. Prepare cron task on every client node to start downloading files at the same time

B. Experiment Results

We ran the experiment several times. Unfortunately it always finished with exceeding the network quota on one server nodes. We discovered that it was because CoMon service did not collect network data which is run as root. As Lighttpd is a server application it is running on special account.

This caused that in our database all server nodes have same weight. So our redirection application always chooses the first of the best ones. Owing to the fact that all have same weight (zero), it always chooses the first one.

C. Conclusions

CoMon service cannot be used in PlanetLab environment for load-balancing network traffic, because it collects a wrong type of data.

The main advantage of this experiment is that we created and tested tools and techniques which we used in following parts. We have built environment for future work.

VI. Experiment 2

A. Environment Modifications

Service Registry

We did not change the architecture on root server. The only modification was shell script which collects data from Web Servers. We collected TX rates extracted from Web Servers network interfaces instead of data returned from CoMon service.

Web Server

We created shell script which retrieves TX rate from ETH0 interface of Web Server. Data generated from this script serves as simple HTML page by PHP script. This PHP script is used by Service Registry to extract TX rate from Web Server

Database

Our database has to grow, because we needed to store all information about TX rates. We decided to create a stored procedure for updating weight for server.

B. Experiment Results

We need to find metric to compare results. Our first idea was to use throughput. But the question was which one: the whole server throughput or just generated by our clients. Calculating metric using the whole server TX rate does not work, because it is not comparable on virtual environment. Especially that we do not know infrastructure behind it. Unfortunately calculating throughput generated only by our clients is also incorrect, because lot of throughput is generated on other virtual machines.

We decided to check how requests were distributed to servers. Having recalculated every weight, we ordered servers by weight and calculated how many requests went to the servers from the one with the least weight up to this with the most weight. The following table (table 1) presents result series by average where $Weight_{i+1} > Weight_{i}$ in the moment of redirection:

Table I.
Results of the experiment 2

Server weight	No. of requests	Percentage of total
Weight1	299	17,798%
Weight2	286	17,024%
Weight3	274	16,310%
Weight4	217	12,917%
Weight5	263	15,655%
Weight6	187	11,131%
Weight7	154	9,167%

C. Conclusion

In this experiment, weights of each Web Server were updated every minute. One minute looks a good factor for redirection approach, especially that updating Web Servers more frequently could start be an important part of throughput. Such approach caused that all requests which were handled by CDN network between recalculations were redirected to one Web Server.

These problems are especially visible in the experiment results. On the next iteration we are trying to improve http request forwarding (i.e. weight recalculation algorithm) to eliminate such side effects.

VII. Experiment 3

A. Environment modifications

Database

As we have described above in §VI.C, our forwarding algorithm needed some changes to be more effective. We tried

to introduce a simple approximation of Web Server's weight between recalculations. Basing on previous weights and number of clients which were handled by Web Server between recalculations, after each user request handled by the Web Server we increase the weight by following factor:

$$\frac{last\ increate\ between\ recalculations}{number\ of\ clients\ between\ recalculations} \quad (1)$$

B. Experiment Results

Using the same metric as in previous experiment, the table with results presents as follows. Again this is results set by average:

TABLE II.
RESULTS OF THE EXPERIMENT 3

Server weight	No. of requests	Percentage of total
Weight1	922	34,17%
Weight2	582	21,57%
Weight3	447	16,57%
Weight4	385	14,27%
Weight5	362	14,27%

We decided to include one more metric: request per server:

TABLE III.
REQUESTS PER SERVER - EXPERIMENT 3

Server weight	No. of requests	Percentage of total
A	435	16,12%
B	760	28,17%
C	534	19,79%
D	483	17,90%
E	486	18,01%

C. Conclusion

Our summary results look very well. 4 out of 5 servers handled a similar number of requests. Moreover two servers with smaller load took most of new incoming requests. Unfortunately we have observed one problem, which is not visible in the summary results. After each recalculation our algorithm completely reorders servers, so the most loaded server starts to be least one. In our opinion it can perturb balance of servers .

VIII. EXPERIMENT 4

A. Environment Modifications

Database

Our previous methodology gave us stable results, but the algorithm reorders the server list after each calculation. That is why we decided to introduce random factor in our algorithm. New redirect implementation should ensure that probability that n'th Web Server will handle user request is higher for those Web Servers which have lower weight (have lower workload). Moreover, dependency between probability that n'th Web Server is chosen for client and weight should not be linear, it should be rather similar to 1/x.

To implement such logic we used the following solution.

1. For each Web Server we calculate following value:

$$x_i = \frac{\sum_{k=1}^{n} w_k}{w_i} \quad (2)$$

2. We ordered ascent values computed in previous step:

$$x_j : j = 1..\mathrm{n}\ \forall k, l : \left(k \le l \wedge l \le n \wedge 1 \le k \rightarrow x_k \le x_l\right) \quad (3)$$

3. Based on previously calculated values we evaluated:

$$z_j = \frac{\sum_{m=1}^{j} x_m}{\sum_{k=1}^{n} x_k} \quad (4)$$

Definition of these values shows that following statement is true:

$$max\left(z_j : j = 1...\mathrm{n}\right) = 1 \quad (5)$$

To calculated values, we add additional one:

$$z_0 = 0 \quad (6)$$

4. Having performed the above operations, we have n+1 weights which all are in range [0, 1]. Moreover it can be proof that:

$$\forall k, l : \left(0 < k, l \le n \wedge x_k < x_i \rightarrow z_k - z_{k-1} \ge z_l - z_{l-1}\right) \quad (7)$$

5. In the last step, we randomized a number from range [0,1) and looked for minimal value of zj which is greater than randomized number. Random number is needed to make better distribution between recalculations.

B. Experiment Results

Using the same metric as in previous experiments, the table with average results from series of experiments presents as follows (table 4)

TABLE IV.
RESULTS OF THE EXPERIMENT 4

Server weight	No. of requests	Percentage of total
Weight1	8881	35,17%
Weight2	6432	25,47%
Weight3	4992	19,77%
Weight4	4944	19,58%

Moreover table of requests per server is almost ideal:

TABLE V.
REQUESTS PER SERVER - EXPERIMENT 4

Server weight	No. of requests	Percentage of total
A	7064	27,98%
B	5524	21,88%
C	6427	25,45%
D	6234	24,69%

C. Conclusion

Out last experiment gave us really good results. Servers are well-balnced. The difference between the most loaded and the least loaded is around 6%. Moreover the order list is stable between recalculations and still less-loaded servers take more than 60% of requests.

IX. FINAL CONCLUSIONS

Building a well-balanced CDN one does not need difficult algorithms. Some of them of course are better than others. The most important thing is assumption: every node must be same or very similar to others. If not, balancing function must include differences between nodes.

Our set of experiments presents evolution of an balancing algorithm: from very simple to complex. Moreover we have created a technique which is working on such unpredictable environment as Planetlab. As we have described above Planetlab is a very good Internet simulation.

The presented technique can deal with following problems:

- Infrastructure – every Planetlab node should be connected to the internet with the same 100Mb/s cable. We cannot check every node we use in experiment, but during our internal test at Warsaw University of Technology, we discovered network problems. As in real life we cannot be sure of the Internet speed.
- Virtualization – this is the problem with sharing resources. For example on one physical computer we have several virtual machines. They share CPU, disk speed, physical RAM and the most important in this experiments network card.

Nowadays when we start to use virtual computers more than the real ones, we have to deal with different problems than 10 years ago. The above methodology can be used in every virtual environment. Taking into account that implementation details probably have to be adapted in technical implementation .

REFERENCES

[1] Arlitt, M., Jin, T., Workload characterization of the 1998 world Cup Web. IEEE Network 14; pp 30 -37, 2000.
[2] Pathan, M., Buyya, R., Broberg, J, Internetworking of CDNs, in Content Delivery Networks, Springer-Verlag Berlin Heidelberg, pp 389-413, 2008.
[3] Princeton web page http://comon.cs.princeton.edu/
[4] PSSH project web page http://www.theether.org/pssh/
[5] WGET documentation http://www.gnu.org/software/wget/
[6] Lighttpd documentation http://www.lighttpd.net/
[7] PostgreSQL documentation http://www.postgresql.org/
[8] CRON documentation http://en.wikipedia.org/wiki/Cron