Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Platformově nezávislý masivně paralelní distribuovaný testovací framework

2019

Jan Kašík

Vedoucí práce: Jméno vedoucího práce

Studijní obor: Informatika, prezenční forma

Bibliografické údaje

Autor:	Jan Kašík
Název práce:	Platformově nezávislý masivně paralelní distribuovaný testovací framework
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2019
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	Jméno vedoucího práce
Počet stran:	14
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Jan Kašík
Title:	Platform agnostic massively parallel distributed testing framework
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2019
Study field:	Computer Science, full-time form
Supervisor:	Jméno vedoucího práce
Page count:	14
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Cílem této diplomové práce, vypsané společností Red Hat Czech, s.r.o., je analyzovat možnosti v oblasti masivně paralelního distribuovaného testování a vytvořit prototyp frameworku, který adresuje nedostatky a obchází překážky nalezené v existujících řešeních. Práce se skládá ze dvou částí. V první části student provede podrobnou analýzu zaměřenou na funkční schopnosti, použité algoritmy a inženýrské přístupy použíté v existujících systémech. Ve druhé části student vytvoří prototyp frameworku pro multiplatformní synchronizaci kontextu mezi uzly. Framework bude možné použít k událostmi řízené orchestraci rozmanitých platforem pomocí agentů. Framework bude distribuovat a orchestrovat agenty pomocí kterých bude exekuovat a synchronizovat různé úlohy. K exekuci a obsluze těchto úloh budou použity zásuvné konektory. Framework bude umožňovat uživateli přidat nové zásuvné konektory pro jiné platformy a bude tak rozšiřitelný. Framework bude používat konfigurační jazyk pro abstrakci uživatele od úloh samých (doménově specifický nebo obecný jazyk na uvážení studenta). Úroveň abstrakce poskytovaná zásuvnými konektory bude tak vysoká, že uživatel nebude muset používat nativní jazyk dané platformy. Zdrojový kód práce bude dostupný pod podmínkami licence GNU/GPLv3 nebo kompatibilní.

Synopsis

The goal of this thesis is to analyse open vistas in the field of massively parallel distributed testing and to prototype a framework that addresses shortcomings and overcomes obstacles found in currently existing solutions. The thesis consists of two parts, in the first part, student carries out a thorough analysis focused on operational capabilities, used algorithms and engineering approaches leveraged in currently existing systems. In the second part, student will create a prototype of a framework for multiplatform context synchronization between nodes. It will be possible to use this framework to orchestrate various platforms by its agents in an event driven way. The framework will distribute and orchestrate agents through which it will be executing and synchronizing various tasks. Pluggable connectors will be used to execute and manipulate these tasks. Framework will allow user to create and add additional pluggable connectors for other platforms to keep it extensible. Framework will be using configuration language to abstract user from tasks itselves (DSL or GPL – at discretion of student). The level of abstraction, provided by pluggable connectors, should be so high, that user does not have to use platform's native language. The final thesis source code will be available under terms of GNU/GPLv3 licence or compatible.

Klíčová slova: paralelní a distribuované systémy, testování, open source

Keywords: parallel and distributed systems, testing, open source

Děkuji, děkuji, děkuji.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

1 Motivation **8**
1.1 High availablity . . . 8
1.2 Load balancing . . . 9
1.2.1 DNS load balancer . . . 9
1.2.2 Content delivery network . . . 10
1.3 Everyday life distributed systems . . . 11
1.3.1 Operating-system-level virtualization . . . 11
1.3.2 Cloud . . . 11
1.3.3 Microservices . . . 11
1.4 mod_cluster . . . 11

2 The output of my work **12**

3 Historical solution for distributed testing **13**

4 Contemporary solutions for distributed testing **13**
4.1 Radar gun . . . 13

5 The problem **13**

6 My contribution/the solution **13**
6.1 Openshift/Kubernetes automation . . . 13
6.2 Traffic control . . . 13
6.3 Automate it/putting it together . . . 13

7 Implementation part **13**

Literatura **14**

Seznam obrázků

Seznam tabulek

1 Ilustration of system downtimes based on availability agreed in SLA 9
2 Comparsion of various load balancers 12

Seznam vět

Seznam zdrojových kódů

1 Answer section of output from dig upol.cz command 10
2 Answer section of output from dig inf.upol.cz command 10

Upozornění: Následující text je rozpracovaná a (značně) neúplná verze!!!

1 Motivation

Everyday we enjoy perks of distributed systems. From life like examples like network of independent railway stations with trains as a mean of connection between them to practical computer solutions which are build for various reasons. Of which the scalability, high availability and security would be the most frequent.

I will try to bring some of those use cases closer to reader to show, how important distributed systems are and how crucial is to verify that they are working properly.

1.1 High availablity

All users of any service want to have an access to it everytime they wish. Failure of a critical service which would cause its inaccessibility when you need it is not an acceptable scenario in most cases. This implies that we need to know how reliable some service is if we want deploy it to some business critical environment. The degree of this reliability is also called availability.

The simplest definition of average avalability A is proportion of expected uptime (mean time between failures) of some service to sum of expected downtime (mean time to restore) and uptime

$$A = \frac{E(uptime)}{E(uptime) + E(downtime)}$$

this gives us a percentage of time during which system performs its required function[1]. In this case it is a time during which the service is available to users.

Provider and user usually agreeds on SLA (service level agreement). There are also specified other aspects of service like quality and responsibilities of contracting parties. Table 1 shows, how agreed availability varies from values common for cheap web hosting service to critical availability typical for realm time systems. Such promised availability is also called **high availability**.

To achieve such availability, multiple approaches are being used. The main one is redundancy. Redundancy in harware is the easiest way how to achieve one. Redundant power supply, redundant storage (e.g. RAID) etc. It just won't help when that one machine blows its single point of failure which is usually mother board. In that case it is clear we need multiple machines working as service providers. Nodes, which would run the same code, providing the exact same service.

There is now multiple nodes providing same service, but how to control to which one users connect without breaking their comfort? We need a top layer

Availability	MMTR/year	MTTR/month	MTTR/week	MTTR/day
90	36.5 d	72 h	16.8 h	2.4 h
99	3.65 d	7.2 h	1.68 h	14.4 m
99.5	1.83 d	3.6 h	50.4 m	7.2 m
99.9	8.76 h	43.8 m	10.1 m	1.44 m
99.99	52.56 m	4.38 m	1.01 m	8.64 s
99.999	5.26 m	25.9 s	6.05 s	864.3 ms
99.9999	31.5 s	2.59 s	604.8 ms	86.4 ms
99.99999	3.15 s	262.97 ms	60.48 ms	8.64 ms
99.999999	315.569 ms	26.297 ms	6.048 ms	0.864 ms
99.9999999	31.5569 ms	2.6297 ms	0.6048 ms	0.0864 ms

Tabulka 1: Ilustration of system downtimes based on availability agreed in SLA

mechanism in infrastructure to take the control over "redirection"of users requests (which is very simplified) and detecting failures in lower layer. Although this mechanism becomes new single point of failure and evade this may be challenging, it makes the whole system more reliable.

1.2 Load balancing

This kind of mechanism is called **load balancer**, since it distributes workload between redundant nodes in cluster. It must be also said that this load balancer needs to detect failures of individual nodes and stop redirecting to those which fail. Verifying funcionality of such load balancer is very complex and is main motivation for my thesis. I will now introduce several examples of load balancers.

1.2.1 DNS load balancer

The simplest way of load balancing is DNS based load balancing. If you are familiar with Domain Name Service, you can skip following two paragraphs which introduce reader into the matter.

The main purpose of DNS is to associate IP address with a hostname. A hostname, which is far more rememberable for human being than, in a worst case, a sequence of hexadecimal symbols divided by colons. When user initites a basic command `ping` with a hostname as an argument, a program called `resolver` takes over. For the purpose of an example, all caches will be omitted and the hostname will be `upol.cz`. The `resolver` first asks DNS server configured in operating systems, if it knows the IP address of `upol.cz`. It probably doesn't, but it provides resolver with an address of someone who could – the root server. `resolver` asks root server for the address, but it doesn't know it. It only knows who could know it – the server responsible for `cz` domain and provides resolver with its address. It sends query to that server once again, asking, who is reponsible for `upol.cz` domain. After receiving an answer, the `resolver` sends a final query to that address recieving response like shown in source code 1.

```
;; ANSWER SECTION:
upol.cz.  86022 IN A 158.194.230.95
```

Zdrojový kód 1: Answer section of output from dig upol.cz command

Note that the response in source code one provides just one address for `upol.cz` domain, since this will be subject of further explanation. This is just a short introduction to DNS and it definitely doesn't cover all shortcomings and corner cases.

DNS load balancing is round-robin based. With appropriate DNS server configuration, the answer recieved by `resolver` looks like the one in source code 2. There are multiple IP addresses assigned for one hostname. The resolver usually picks the first and the DNS server rotates these addresses in every response so the traffic is equally distributed.

```
;; ANSWER SECTION:
inf.upol.cz.  86400 IN A 158.194.92.6
inf.upol.cz.  86400 IN A 158.194.80.18
```

Zdrojový kód 2: Answer section of output from dig inf.upol.cz command

Although the principle of this simple system sounds great, it really doesn't work so well. The main issue is caching. In DNS, almost everything is cached with various expiration time anywhere between few days to a week. This causes the fact, that lot of clients which queries DNS server with cached queries within expiration interval, recieve same response. Thus they will try to connect to the same server which can put a significant load on it.

This solution also cannot detect any failures of servers which addresses it provides. There is no protocol for a talk between DNS server and host, so there is no way for DNS server how to find out that the host is not alive anymore [2][3].

1.2.2 Content delivery network

Important factor which influences response time of target host is its geographic location in relation to client. CDN or more archaical term Global server load balancing[2] relies on fact, that the closer the host is to the client, the faster is response from the host from the client's point of view. Not only response can be faster. If the host and client are in the same wider network, the client can usually access to the host on a connection with higher bandwidth, thus downloading content faster[3].

From historical point of view, in times, when connection was priced by bytes transfered, the internet service providers (ISPs) tried to find a way how to save money by transfering less data through foreign networks. The solution was quite

simple and efficient. The ISPs established a mirror (usually a FTP server) of often downloaded large files from those networks and provided it to its clients. This was beneficial for both sides. ISPs saved money and client downloaded desired file faster.

As it was already said, the basis of this system stands on servers in various geographic locations which mirrors their content. As of present, you can pay for service proving funcionality of CDN and use it for static content of your website since building network like this can be and in fact is very expensive.

1.3 Everyday life distributed systems

Although verifying the correct funcionality of a load balancer is very important, there are other use cases when confirming precise funcionality of a distributed system is crucial. Such systems are more and more common since they are much easier to scale. In distributed system, when more power is needed, the only thing which has to be done is add an other new node.

1.3.1 Operating-system-level virtualization

So called containers or less likely heard words written in the heading are isolated instances of operating systems kernels running in user-space. Due to the fact that for the programs running inside them is their environment almost unrecognizable from real environment, are such containers used commonly for virtualization hosting. When comes to security, containers comes handy to, because it is possible to isolate applications in them too.

1.3.2 Cloud

Cloud computing based on lending out the server resources is another popular term nowadays.

1.3.3 Microservices

Running pieces of code which are loosely connected between each other. A server oriented architecture which promotes writing applications how they should be really written – modularized, independent and reusable chunks.

1.4 mod_cluster

The initial comes for this thesis comes from need to test mod_cluster – a HTTP based load balancer. As I made introduction to load balanciong before, now I will only make few notes about mod_cluster and its versaility.

First, let's introduce some neccassary terminology. Client in following text is an end user, who wants to profit from working load balancing. Balancer is a network node which performs load balancing. Worker is a network node which

Tabulka 2: Comparsion of various load balancers

	NGINX	mod_cluster	mod_jk	HAproxy
Dynamic worker configuration	no[5]	yes[4]	no	
Load balancer factor provided by worker	no[5]	yes[4]	N/A[1]	
Application life cycle control	no[2][5]	yes[4]	no	

consumes client's requests and generates responses – often an application server with deployed application.

mod_cluster uses own set of HTTP methods, often called mod_cluster management protocol (MCMP)[4], to establish communication between workers and balancer. This channel can be used to transmit load balance factors and application lifecycle events from workers to balancer.

Over other HTTP based load balancers, mod_cluster has following advantages:

Application life cycle control. Most of other solutions cannot distinguish between request for undeployed application from application server and request for non-existent resource. In both cases, this kind of request ends up with a 404 HTTP response code. In mod_cluster load balancing environment, workers forwards to balancer application lifecycle events like deploy and undeploy. The balancer then know when it can stop or start routing requests to that worker.[4]

Worker side load balance factor calculation. Traditional HTTP based solution calculate load balance factors on balancer on balancer node. In mod_cluster environment, these factors are calculated and provided by worker nodes.[4]

Dynamic worker configuration.

2 The output of my work

==========

[2]There is no load balancer factor in mod_jk load balancing

[2]It however looks for server's health[5]

3 Historical solution for distributed testing

4 Contemporary solutions for distributed testing

4.1 Radar gun

5 The problem

First performance then float to degrading network testing

6 My contribution/the solution

6.1 Openshift/Kubernetes automation

sunstone, jclouds, xft-cz

6.2 Traffic control

creating degraded network in Openshift

6.3 Automate it/putting it together

7 Implementation part

Literatura

[1] BIROLINI, Alessandro. *Reliability Engineering: Theory and Practice*. Seventh edition. Tuscany, Italy: Springer, 2014. ISBN 978-3-642-39535-2.

[2] BOURKE, Tony. *Server load balancing*. Sebastopol, CA: O'Reilly, 2001. ISBN 0596000502.

[3] MEMBREY, Peter. *Practical load balancing : ride the performance tiger*. Berkeley, CA New York: Apress Distributed to the Book trade worldwide by Springer, 2012. ISBN 9781430236818.

[4] JBOSS.ORG. *About mod_cluster* [online]. 2017 [cit. 2018-2-10]. Dostupný z: ⟨http://modcluster.io⟩.

[5] NGINX. *NGINX Load Balancing - HTTP and TCP Load Balancer* [online]. 2017 [cit. 2018-2-10]. Dostupný z: ⟨https://www.nginx.com/resources/admin-guide/load-balancer/⟩.