Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Platformově nezávislý masivně paralelní distribuovaný testovací framework

2017

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:	2017
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	Jméno vedoucího práce
Počet stran:	11
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:	2017
Study field:	Computer Science, full-time form
Supervisor:	Jméno vedoucího práce
Page count:	11
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 použité 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.

Obsah

Seznam obrázků

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

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.

I will give an example of such load balancers. The simplest way of load balancing is DNS based load balancing. When DNS server responds to client with an IP address of target server, the DNS server usually sends a list of IP addresses and client chooses the first. This method relies on rotating those addresses on round-robin base, so each different client will use different IP address to connect to the server. This way has one major issue. It cannot detect any failures of a network or a server. The second problem is, that the DNS reponds are cached. That means, that a lot of clients eventually will use the same IP address, since order of addresses in list won't change.

Another common example of load balancer is FastCGI based web servers. Such servers run several reusable instances of same CGI script and it is decided to which instance will be a request send.

And last but not least is HTTP load balancer. Verifying funcionality of such load balancer is very complex and is main motivation for my thesis.

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.

2 The output of my work

Literatura

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