A Use-Case on Testing Adaptive Admission Control and Resource Allocation Algorithms on the Federated Environment of Panlab

Christos Tranoris, Pierpaolo Giacomin, and Spyros Denazis

Electrical and Computer Engineering department, University of Patras,
Rio, Patras 26500, Greece
tranoris@ece.upatras.gr, yrz@anche.no, sdena@upatras.gr

Abstract. Panlab is a Future Internet initiative which integrates distributed facilities in a federated manner. Panlab framework provides the infrastructure and architectural components that enable testing applications near production environments over a heterogeneous pool of resources. This paper presents a use case where an adaptive resource allocation algorithm was tested utilizing Panlab's infrastructure. Implementation details are given in terms of building a RUBiS testbed that provides all the required resources. Moreover, this experiment needs to directly request, monitor and manage resources that it uses during the experiment. As a result of this use case a new feature for Panlab was developed called Federation Computing Interface (FCI) API which enables applications to access resources during an experiment.

Keywords: Panlab, experimental testing, resource federation, Future Internet

1 Introduction

Future Internet research results in new experimental infrastructures for supporting approaches that exploit extend or redesign current Internet architecture and protocols. The Pan-European laboratory [1], Panlab, is a FIRE[2] initiative and builds on a federation of interconnected and distributed facilities allowing third parties to access a wide variety of resources like platforms, networks, and services for broad testing and experimentation purposes. In this context, Panlab defines a provisioning framework and a meta-architecture that give rise to a number of Federation Mechanisms and Architecture Elements to be used for experimentation in the Future Internet.

The Panlab infrastructure manages interconnections of different geographically distributed testbeds to provide services to customers for various kinds of testing scenarios which in Panlab terminology are called Virtual Customer Testbeds or simply VCTs. A VCT is a specification of required (heterogeneous) resources along with their configurations, offered by a diverse pool of organizations in order to form new richer infrastructures. These VCTs represent customer needs such as i) evaluation and testing specifications of new technologies, products, services, ii) execution of network and application layer experiments, or even iii) complete commercial applications that are executed by the federation's infrastructure in a cost-effective way.

Panlab's architecture introduces components for integrating testbeds that belong to various administrative domains, in order to become available to participate in testing scenarios. A Web Portal is available where customers and providers can access services, a visual Creation Environment which is called "Virtual Customer Testbed (VCT) tool" where a customer can define requested services, a repository which keeps all persistent information like resources, partners, defined VCTs, etc. Experimenters can browse through the resource registry content and can select, configure, deploy and access reserved resources. Finally, an Orchestration Engine is responsible for orchestrating the provisioning of the requested services. The above components interact with each other in order to offer a service called "Teagle". Part of Teagle is also the Teagle Gateway, the component that is responsible for transferring provisioning and configuration commands to selected resources lying in various administrative domains. The functionality of Panlab office is complemented by a Policy engine. All components communicate via an HTTP-based (REpresentation State Transfer) RESTful interface. A per domain central controller is the Panlab Testbed Manager (PTM). PTM is responsible for accepting RESTful commands from the Teagle Gateway in order to configure the domain's resources. PTM implements the so called Resource Adaptation Layer where Panlab partners "plug-in" their Resource Adapters (RA). A Resource Adapter (a concept similar to device drivers) wraps a domain's resource API in order to create a homogeneous API defined by Panlab. Details and specifications of Panlab's components can be found at [1].

This paper describes an experiment made utilizing the Panlab's framework and available infrastructure. The challenge here were twofold: i) to run the experiment by moving a designed algorithm from a simulating environment to near production best-effort environment and ii) to exploit the framework in such a way that will allow the system under test to directly request or release resources that it uses. The latter indicates that the experiment needs access to the whole framework after the provisioning during the operational phase. As a result to accomplish the needs of this experiment was the development of a new feature of Panlab's framework called Federation Computing Interface (FCI) API. FCI enables the access of provisioned/reserved resources during the execution of an experiment.

The rest of this paper is organized as follows: The first section describes the use case requirements and the needed infrastructure. The second section describes the implementation of the infrastructure and the deployment of the testbed. The third section discusses the execution of the experiment and how Panlab framework is able by means of Federation Computing Interface API to managed resource. We finally conclude this paper.

2 Use Case Description

In order for one to test an adaptive admission control and resource allocation algorithm, it is necessary to set up an appropriate testbed of a distributed web application like RUBiS benchmark [3], an auction site prototype modeled after eBay.com. It provides a virtualized distributed application that consists of three components, a web server, an application server, a database and a workload generator, which produces the appropriate requests. Furthermore it can be deployed in a virtualized environment using Xen server technology, which allows regulating system resources such as CPU usage and memory, and provides also a monitoring tool, Ganglia, that measures network metrics, such as round trip time and other statistics, and resource usage in virtual machines.

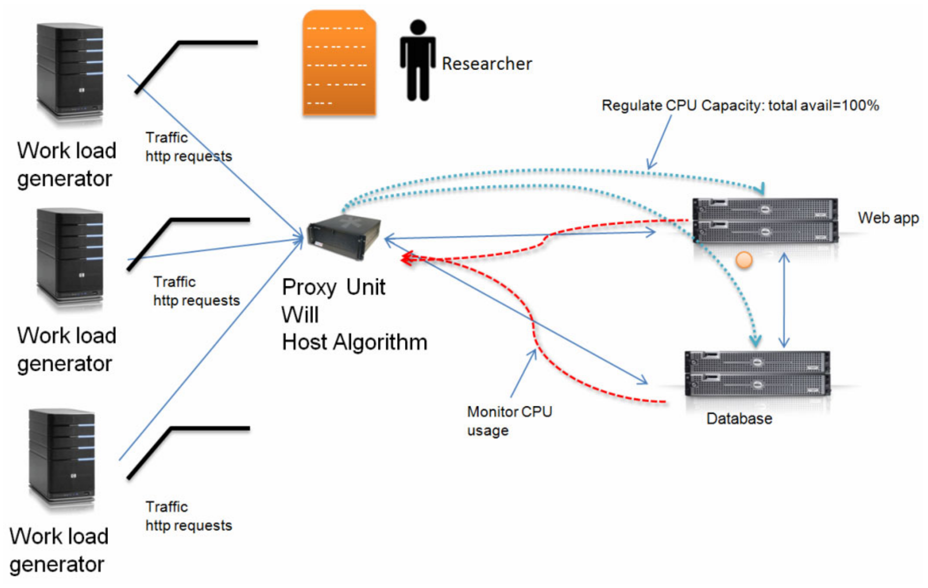


Fig. 1. The setup for testing the algorithm

The adaptive admission control and resource allocation algorithm is applied to succeed in specific target of network metrics, like round trip time and throughput. This will be done by deploying a proxy-like control component for admission control and using Xen server technology to regulate CPU usage. During this scenario the adaptive admission control and resource allocation algorithm is tested against network metrics, like round trip time and throughput. RUBiS clients will produce requests so that push RUBiS components to their limits, so that resource like CPU usage and network throughput get high values.

During the setup, the researcher wants to test http proxy software written in C programming language that implements an admission algorithm. Figure 1 displays the setup for the discussed scenario. The setup consists of 3 work load http traffic generators, making requests through a hosting unit. The algorithm, which is located at the proxy unit, needs to monitor the CPU usage of the Web application and Database machines. Then the algorithm should be able to set new CPU capacity limits on both resources. Additionally the algorithm should be able to start and stop the work load generators on demand.

3 Technical Environment, Testbed Implementation and Deployment

From the requirements of the use case, it is evident that it would benefit from a testbed offering RUBiS resources. Moreover, the experiment needs to manage and monitor resources within the C algorithm. So the resources need to provide monitoring and provisioning mechanisms.

To support such an experiment and similar ones, a required infrastructure needed to be built. The equipment used is as follows:

- Linux machines for the RUBiS based work load generators
- A Linux machine for the hosting the algorithm unit, capable of compiling C and Java software
- Linux machines for running XEN server where on top will run the RUBiS Web app and database

The final user needs to provide the algorithm under test. He will just login to the Proxy Unit, compile the software and execute it. The user will not have access to the RUBiS resources (i.e. cannot login) so there is a need to encapsulate the monitoring and provisioning capabilities. For this requirement and to make available the RUBiS resources for future testing within the Panlab federation, the so called Resource Adapters (RA) where built.

For each resource there is a corresponding RA which exposes configuration parameters to the end user. As displayed in Figure 2, all the components are based on Virtual Machines managed by a XEN server. The implemented RAs instantiate all these Virtual Machines and configure the internal components according to end-user needs. The work load generator exposes parameters such as: used IP for the testbed, memory, hard disk size, number of clients, ramp up time for the requests and a parameter used during the execution of the experiment called Action which accepts the values start and stop. The Proxy Unit exposes parameters such as used IP for the testbed, memory, hard disk size, username, password and IP to connect to the RUBiS application resource. The RUBiS application and the RUBiS database have similar parameters to the above and additionally a MON_CPU_UTILIZATION parameter which is used to monitor the resource and a CPU_CAPACITY used to set the max cpu capacity of the resource.

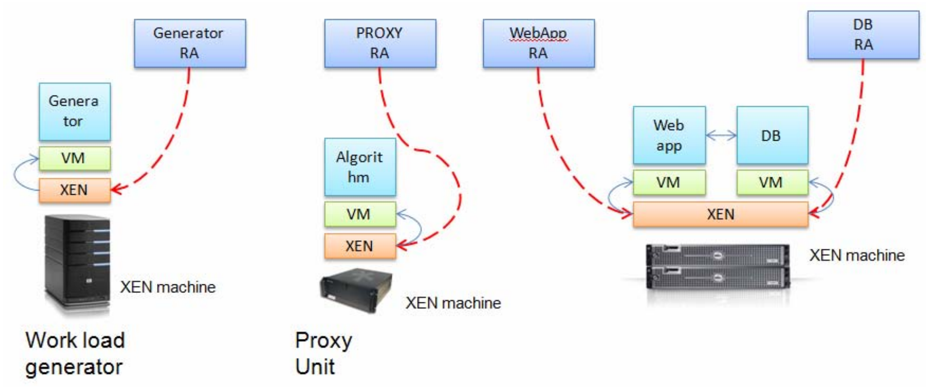
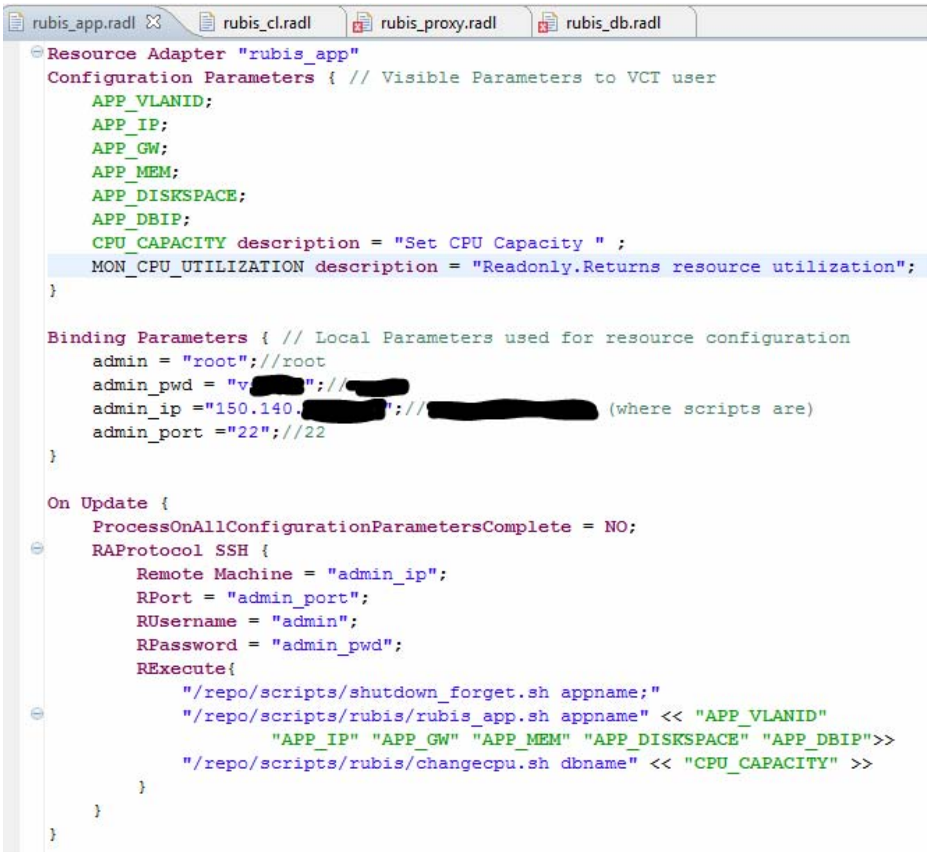


Fig. 2. The Resource adapters of the available testbed resources



```
Resource Adapter "rubis_app"
Configuration Parameters { // Visible Parameters to VCT user
    APP_VLANID;
    APP_IP;
    APP_GW;
    APP_MEM;
    APP_DISKSPACE;
    APP_DBIP;
    CPU_CAPACITY description = "Set CPU Capacity " ;
    MON_CPU_UTILIZATION description = "Readonly.Returns resource utilization";
}

Binding Parameters { // Local Parameters used for resource configuration
    admin = "root";//root
    admin_pwd = "v[redacted]";//[redacted]
    admin_ip ="150.140.[redacted]";//[redacted] (where scripts are)
    admin_port ="22";//22
}

On Update {
    ProcessOnAllConfigurationParametersComplete = NO;
    RAProtocol SSH {
        Remote Machine = "admin_ip";
        RPort = "admin_port";
        RUsername = "admin";
        RPassword = "admin_pwd";
        RExecute{
            "/repo/scripts/shutdown_forget.sh appname;"
            "/repo/scripts/rubis/rubis_app.sh appname" << "APP_VLANID"
                "APP_IP" "APP_GW" "APP_MEM" "APP_DISKSPACE" "APP_DBIP">>
            "/repo/scripts/rubis/changecpu.sh dbname" << "CPU_CAPACITY" >>
        }
    }
}
```

Fig. 3. RADL definition for the RUBiS application resource

The resource adapters where defined using the Panlab's Resource Adapter Description Language (RADL)[4]. RADL is a concrete textual syntax for describing a Resource Adapter based on an abstract syntax defined in a meta-model. RADL is an attempt for describing a RA in a way that decouples it from the underlying implementation code. RADL's textual syntax aims to be easier to describe a RA than code in Java or other target language. RADL is useful in cases when there is a need to configure a resource that offers an API for configuration. The user can configure the resource through some Configuration Parameters. The RA "wraps" the parameters and together with the Binding Parameters, the RA can configure the resource. A Binding Parameter is a variable that is assigned locally by the resource provider, e.g. a local IP address. This approach was also adopted for developing the RUBiS RAs. Figure 3 displays the RADL definition for the RUBiS application server.

The Configuration Parameters section describes the exposed parameters to the end user. The Binding Parameters are used for internal purposes of the local testbed configuration. The On Update section describes what the rubis_app RA does when it receives a provisioning update command from the upper layers. The RA will use the SSH protocol to connect to the internal machine and execute scripts on it.

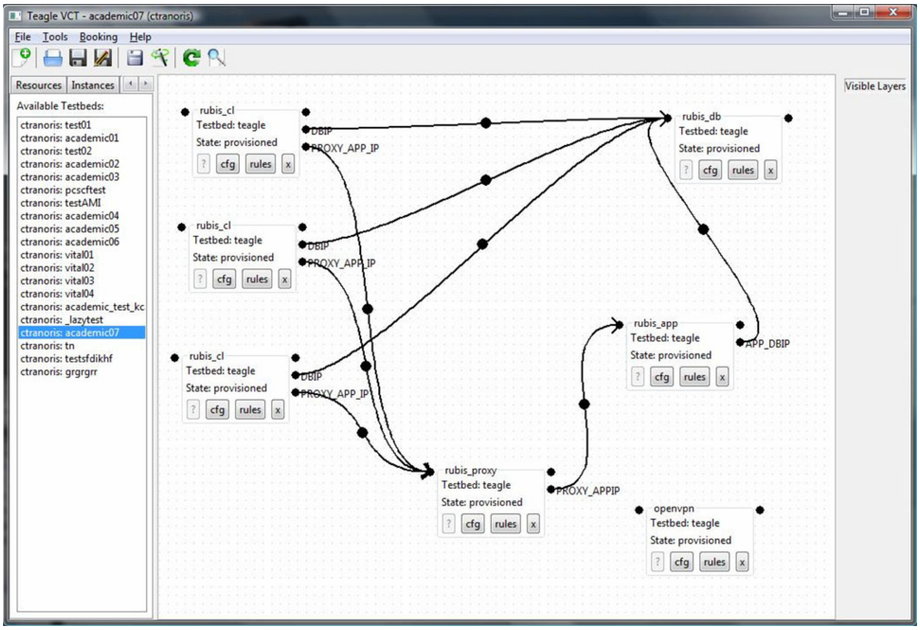


Fig. 4. The RUBiS use case setup designed in the VCT tool

The tools to deploy, monitor and run the experiments are those offered by the Panlab architecture [1]. The Resource Adapters where loaded in the testbed PTM and made available through Panlab's repository. Figure 4 displays the use case setup as can be done inside the VCT tool of Panlab. The resources are available in the left side of the tool. Three rubis client where selected one rubis proxy, one rubis application and one rubis database. Interconnections where made also between these components in order to assign reference values to all resources. For example the RUBiS clients need to know about the IP of the proxy which hosts the algorithm. The proxy needs to know the IP of the RUBiS application which also needs a reference to the RUBiS database.

4 Running and Operating the Experiment

The scenario during the experiment utilizes the Federation Computing Interface (FCI) API that Panlab provides [5]. Federation Computing Interface (FCI) is an API for accessing resources of the federation. It is an SDK for developing applications that access VCT requested resources through the Panlab office services during operation of testing. It is quite easy to embed it into your application/ SUT in order to gain control of the requested resources during testing. The FCI is delivered to the customer after the generation of the SLA and not only does it contain the necessary libraries but also the alias of the resources that are used in the VCT scenario. This allows the User-Application/SUT to access the testbed resources during execution of the experiment in order to manage and configure various environment parameters or to get status of the resources.

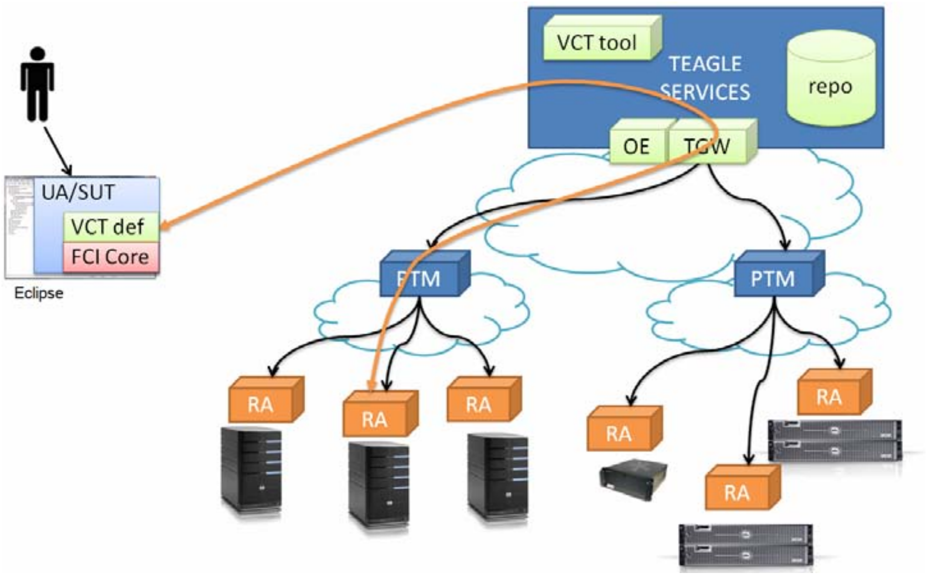


Fig. 5. Designing the algorithm to operate resources during execution

In our testing scenario there is a need to configure resources or even get monitoring status data properly after the VCT is provisioned and while the testing is in progress. Figure 5 displays this condition where the System Under Test (SUT) is our algorithm. FCI automatically creates all the necessary code that the end user can then inject inside the algorithm's code. The end-user needs just to ender his credentials in order

FCI to generate the necessary wrapper classes and functions that are capable of accessing the reserved, provisioned resources. An example is given in the following code listing in Java:

```
//an example Java federation program
public class Main {
  public static void main(String[] args) {
    //An example for VCT: academic07
  academic07 myvct = new academic07();
  myvct.getuop_rubis_cl_91().setRAMP_UP_TIME("55000");
  myvct.getuop_rubis_cl_91().setACTION( "start" );
  myvct.getuop_rubis_cl_91().setNUM_CLIENTS( "300" );
  int moncpu =
myvct.getuop_rubis_db_33().getMON_CPU_UTILIZATION();
```

Assuming that we have given the name academic07 for our VCT definition, the java listing displays how we can access the resources of this VCT. FCI creates a java class, called academic07() that we can instantiate in order to get access to the resources. Additionally, for each resource that participates in the VCT java classes are able to provide access. For example the command `myvct.getuop_rubis_cl_91().setACTION( "start" );` starts the RUBiS client of the rubis_cl_91 resource. The command myvct.getuop_rubis_db_33().getMON_CPU_UTILIZATION(); is able to give back the CPU usage of the database resource.

5 Conclusions

The results of running an experiment in Panlab are encouraging in terms of moving the designed algorithms from simulating environments to near production environments. What is really attractive is that such algorithms can be tested in a best-effort environment with real connectivity issues that cannot be easily performed in simulation environments. The presented use case example demonstrated the usage of existing experimental facilities in this case by exploiting the Panlab framework. The interesting of this experiment is that it extends the framework to allow the system under test to directly request or release resources that it uses.

5.1 Results

First results of running such an experiment although not comparable currently with similar approaches are really encouraging in terms of moving the designed algorithms from simulating environments to near production environments. Using the existing deployed RUBiS facility makes the setup and scaling up of such a testbed much easier. What is really attractive is that such algorithms can be tested in a best-effort environment with real connectivity issues that cannot be easily performed in simulation environments. The scenario presented can be easily scaled up with many clients and web applications. Also, the proxy under test can be replaced by one or more load balancers.

5.2 Testbed Availability

The resources for creating similar scenarios are going to be available under the Panlab Office offerings. Currently there are a limited amount of resources that are capable of hosting the RUBiS environment. We expect to make more resources available as demand increases.

Acknowledgments. The work presented in this paper has been performed during PII a Seventh Framework Program (FP7) project funded by EU.

References

1. Website of Panlab and PII European projects, supported by the European Commission in its both framework programmes FP6 (2001-2006) and FP7 (2007-2013): http://www.panlab.net
2. European Commission, FIRE website: Last cited: November 21, 2010, http://cordis.europa.eu/fp7/ict/fire
3. RUBiS, http://rubis.ow2.org/
4. RADL, http://trac.panlab.net/trac/wiki/RADL
5. Federation Computing Interface(FCI), http://trac.panlab.net/trac/wiki/FCI