UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
COMPUTER SCIENCE BACHELOR THESIS

CAROLINA PEREIRA NOGUEIRA

Benchmark Management of Option Pricer Implementations with Hardware Acceleration Based on the Heston Model

Prof. Dr. Lisandro Zambenedetti Granville
Advisor

Dipl.-Ing. Christian de Schryver
Coadvisor

Kaiserslautern, July 2014

"You see things; and you say, 'Why?' But I dream things that never were; and I say, "Why not?"'

— George Bernard Shaw

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Professor Dr. Lisandro Zambenedetti Granville and Dipl.-Ing. Christian de Schryver, my research supervisors, for their patient guidance, enthusiastic encouragement and useful critiques of this research work. I would also like to thank Prof. Dr.-Ing. Norbert Wehn, for receiving students from the Federal University of Rio Grande do Sul (UFRGS) in the microelectronics research group of the Technical University of Kaiserslautern (TU-KL). My grateful thanks are also extended to Prof. Dr. Taisy Weber by providing the opportunity to participate on the exchange program between UFRGS and TU-KL.

I am also grateful for my colleagues and professors, who always encouraged me to strive for knowledge and attempt new challenges. I would also like to extend my thanks to the International School for Graduate Studies (ISGS) of TU-KL for supporting all the students on arriving and helping to overcome bureaucratic issues. In addition, I appreciate Dipl.-Ing. Thomas Fehmel suggestion on writing methods and text structure, as well as MSc. Marcelo Antonio Marotta for his technical advices and suggestions.

Last but not the least important, I owe more than thanks to my family for their support and encouragement throughout my study. Specially to my parents for always being an example of kindness and perseverance.

ABSTRACT

Modern financial market and its institutions currently have a need for faster computation to asset price, specially options. To achieve fair prices, it is necessary to create a realistic market model and this requires complex methods. Frequently higher performance is strongly related with energy consumption. Hardware acceleration algorithms can perform faster computation with less energy consumption. Benchmarks can compare the results of different implementations to help during the decision process of which solution better attend the current demand. On this context, the challenge of how to integrate, communicate and perform benchmarks on heterogeneous hardware emerge. In this work, an integrated benchmark tool is presented using a representational state transfer (ReST) architecture to overcome this problem. Lower network and processing overhead compared to the existing implementation is aimed. Using a Model-View-Controller model, our platform has strong cohesion and modularity which provides flexibility and scalability for the system.

Keywords: Benchmark, hardware acceleration, heuston model, management, monte carlo, network, option pricer, ReST.

Gerenciamento de benchmarks para precificação de opções financeiras com aceleração em hardware baseado no modelo Heston

RESUMO

Atualmente o mercado financeiro moderno e suas instituições tem a necessidade de computação eficiente do preço dos ativos, especialmente opções. Para atingir preços justos é preciso gerar um modelo de mercado realístico. Esse processo demanda métodos complexos. Frequentemente alto desempenho está diretamente relacionado com consumo de energia. Algoritmos de aceleração em hardware podem obter alta performance com um consumo menor energético. Para ajudar no processo decisivo sobre qual solução melhor atende a demanda atual, benchmarks permitem comparar resultados de diferentes implementações. Nesse contexto surge o desafio de como integrar, comunicar e executar benchmark em hardwares heterogêneos. Nós desenvolvemos uma ferramenta integrada de benchmark usando uma arquitetura de transferência de estado representativo (ReST) para resolver esse problema. Nosso objetivo era obter um sistema com uma sobrecarga de rede e de processamento reduzido em comparação com a plataforma existente. A alta coesão e modularidade do sistema é obtida através do modelo model-view-controller, que introduz flexibilidade e escalabilidade no sistema.

Palavras-chave: benchmark, aceleração em hardware, gerenciamento, monte carlo, redes, precificação de opções financeiras, ReST.

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
CRUD	Create, Read, Update, Delete
DAL	Database Abstraction Layer
DOM	Document Object Model
GWT	GWT Web Toolkit
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
MIB	Management Information Base
MVC	Model View Controller
OID	Object Identifier
ReST	Representational State Transfer
RFC	Requests for Comments
ROA	Resource Oriented Architecture
RPC	Remote Procedure Call
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

LIST OF FIGURES

LIST OF TABLES

LISTINGS

CONTENTS

1 INTRODUCTION

On the modern financial market, asset prices change within several milliseconds, along with all related financial product prices. The demand for fast and accurate product pricing and risk computation is increasing (SCHMERKEN, 2011), as well as the need to save energy resources (SCHRYVER; TORRUELLA; WEHN, 2013). Numerical approaches are mandatory for some financial products, such as option pricers, since there is a lack of closed-form solutions for them(SCHRYVER et al., 2011). These methods have high computational complexity which leads to high energy consumption.

To achieve fast execution of market simulations, researchers have been working on hardware acceleration algorithms (SCHRYVER et al., 2011) (JIN; LUK; THOMAS, 2011). However, that may not be a feasible solution if an algorithm wastes a lot of additional energy just to finish few milliseconds faster than a previous one. It is necessary to observe the compensation curve to decide in which situation the replacement of algorithms is worthwhile. To compare solutions, benchmarking is a powerful tool. A single benchmark platform allows to see many simulation results comparatively and integrates sets of tests for different metrics. Observing the results, it is possible to assert which solution is better used in different contexts. Standardized benchmarks are already widely accepted for performance evaluation in many domains, wherefore researchers also want to introduce it for financial mathematics problems (SCHRYVER et al., 2011).

Some challenges arise when trying to integrate heterogeneous hardware with different implementations of algorithms on the same system. The platform itself should not have an impact on the final result. Some requirements are necessary to achieve this goal, such as minimum network and processing overhead. Furthermore, the benchmark tool should also be as flexible as possible to permit the integration of new implementations with minimal effort as well as supporting many users at same time.

There are few published works related to benchmark integration of financial market simulation on heterogeneous hardware currently. It is an interdisciplinary work that needs communication between people with different background on fields such as Computer Science, Electrical and Information Engineering, Mathematics, and Economics.

A benchmark platform was created on this work to integrate algorithm implementations for different approximation methods with hardware acceleration changing the architectural style of previous problem approaches, reducing coupling and increasing cohesion, which makes it possible to improve the system's scalability.

The overall structure of this study takes the form of six chapters, including this

introductory chapter. This work is composed of four themed chapters. On the next chapter are presented the related work which contributes to achieve the proposed goals. Then, on chapter 3, the terminology and basic concepts to understand the content of this work is presented. The fourth chapter is concerned with the methodology used for this study. Chapter 5 looks at how the implementation methodology defined on previous chapter was implemented. Finally, the conclusion gives a brief summary and critique of the solution, and areas for further research are identified.

2 RELATED WORK

In 2011, the first hardware accelerator for option prices with the Heston model was published (SCHRYVER et al., 2011). The Black-Scholes model was widely used to simulate option market because it provides explicit closed-form solutions for the values of certain (European) call and put options (LAI; SPANIER, 1998). Researches started to use differential methods in order to have a more accurate and general result. Those methods have high complexity and demand more resources to execute in a feasible time. Usually, this is directly related to energy consumption, which is a serious global concern. It is necessary to compare financial option pricers solvers. An integrated benchmark tool interconnecting the implementation is a good solution to centralize the results, comparing performance and energy consumption for further analysis.

There is some work that aims to compare FPGA implementations (JIN; LUK; THOMAS, 2011), but there are many other solvers implemented on distinct hardware types, such as GPU and CPU, for example. The microelectronics research group from the Technical University of Kaiserslautern (EIT/TU-KL), associated with the Institute of Informatics of the Federal University of Rio Grande do Sul (INF-UFRGS) developed a "Web Interface for Accessing Option Pricer Implementations Based on the Heston Model" (TEIXEIRA; SCHRYVER; WIVES, 2013). This is an interface to run benchmarks on different hardware implementations using a client-server architecture. Figure 2.1 shows the components of such implementation and how they communicate with one another. Some parts have the same protocol, but mostly the system components do not have a homogeneous way to exchange information. Also, some of them use proprietary protocols as well: for example, GWT-RPC to connect different system elements. Nonstandard protocols are problematic regarding to changes. If the technology of a component will be modified on future, it is necessary to verify connection and compatibility with other parts of the system. When a standard protocol is used, the probability of having this kind of problems decreases. Regarding those concerns, new requirements arise, such as increase scalability and cohesion, as well as reduced coupling and technology dependence.

Analogous problems were analysed to complement the research basis for this problem due to a lack of domain-specific research fulfilling the mentioned requirements. Mobile hosts are very similar to many of the target devices concerning processing and memory constrains. Some researches related to mobile and to *Internet of Things* management architectures were used, as well as works related to web services performance on mobile hosts.

To catch and process the benchmark results, it is better to use a different device, for example a gateway, so the target device can perform dedicated tasks, avoiding

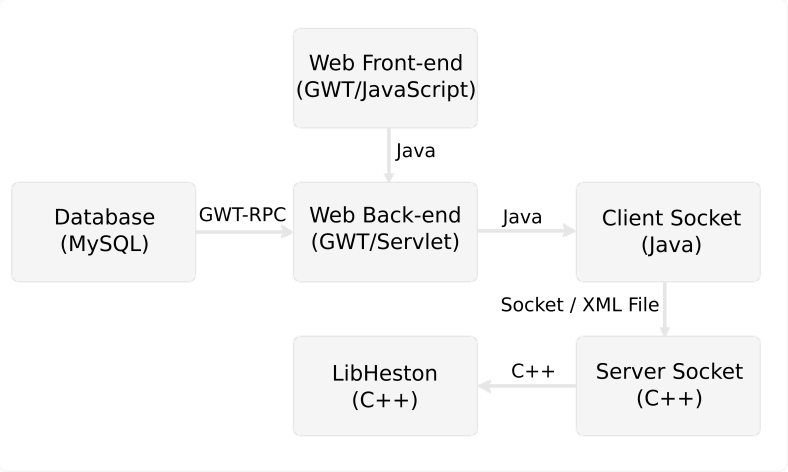

Figure 2.1: Components of "Web Interface for Accessing Option Pricer Implementations Based on the Heston Model" (TEIXEIRA; SCHRYVER; WIVES, 2013)

processing overload and tasks schedulability problems. In addition to that, it is important to pay attention to the network overload, avoiding unnecessary data traffic on the infrastructure. To use a gateway to process the benchmark results, it is necessary to send requests to the target hosts, monitor job status, wait for reply, process the final result, and display it to a human-readable format. This could be considered as a service/network monitoring system because of its structure and functionality. Given the presented reason, this research was based in some of those works (SANCHEZ; ALVARADO-NAVA; MARTINEZ, 2012)(GRANVILLE et al., 2009).

Given the context described before, it is possible to verify the high importance of the comparison between energy consumption versus fast processing of option price solvers. Unfortunately, there is a lack of research work aiming to solve this problem. This affects directly several areas, such as economics, mathematics, electronics, and informatics. This work analyses and presents a solution to perform benchmarks according to the described and previously mentioned constrains and requirements, filling out this gap.

3 BACKGROUND

Some knowledge and concepts are required in order to understand the problem approach. This chapter aims to briefly introduce the necessary information to follow the solution steps trough this work. More explained details of those considerations can be found on the respective references.

3.1 Terminology

3.1.1 Architecture

When the term "architecture" is used along this work, it refers to Fielding's definition: “"An architecture determines how system elements are identified and allocated, how the elements interact to form a system, the amount and granularity of communication needed for interaction, and the interface protocols used for communication."” (FIELDING; TAYLOR, 2002)

3.1.2 Distributed System

A distributed system is a collection of autonomous systems that communicate with one another through a network making use of a distributed middleware. Those proprieties enable system resources sharing and allow them to coordinate their activities. From the user point of view, it results in a single system with integrated computing facility.

3.1.3 Gateway

According to RFC 2616:

> "[A gateway is] a server which acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client may not be aware that it is communicating with a gateway." (FIELDING et al., 1999)

This definition is going to be used along this work.

3.1.4 Service

According to W3:

> "A service is an abstract resource that represents a capability of performing tasks that represents a coherent functionality from the point of view of provider entities and requester entities. To be

> used, a service must be realized by a concrete provider agent." (D. BOOTH H. HAAS, 2004)

It is a resource that performs one or more tasks. Service has a description, an identifier and an interface. Also, it is used by a requester agent and provided by a person or organization.

3.2 Definitions

3.2.1 Document Object Model

> "The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data." (WOOD et al., 1998)

3.2.2 Simple Network Management Protocol

Simple Network Management Protocol (SNMP) is a widely used protocol for network management (STALLINGS, 1999). Most devices support it natively. It is a standard protocol defined by Internet Engineering Task Force (IETF) on the Requests for Comments (RFC) 1157 (CASE et al., 1990). There are many others specific RFCs that refers to different versions of this protocol, each one with their particularities.

3.2.3 Architecture elements of SNMP Management Frameworks

The terms used in this research to refer this architecture's components are according to RFC 3411. It "defines a vocabulary for describing SNMP Management Frameworks, and an architecture for describing the major portions of SNMP Management Frameworks." (HARRINGTON; PRESUHN; WIJNEN, 2002) This kind of architecture has two main entities:

- **Agent**: An SNMP entity containing one or more command responder and/or notification originator applications (along with their associated SNMP engine).
- **Manager**: An SNMP entity containing one or more command generator and/or notification receiver applications (along with their associated SNMP engine).

Management Information Base (MIB) is used by the agent to store the collected data and the response for manager's requests. It is a text file following the Abstract Syntax Notation One (ASN.1) standard. The RFC 1155 (ROSE; MCCLOGHRIE, 1990) defines writing rules of the MIB file in SMI V1 and the RFC 1213 (MCCLOGHRIE; ROSE, 1991), contains the object definition that should be implemented in an agent. Figure 3.1 shows an example of SNMP management framework architecture.

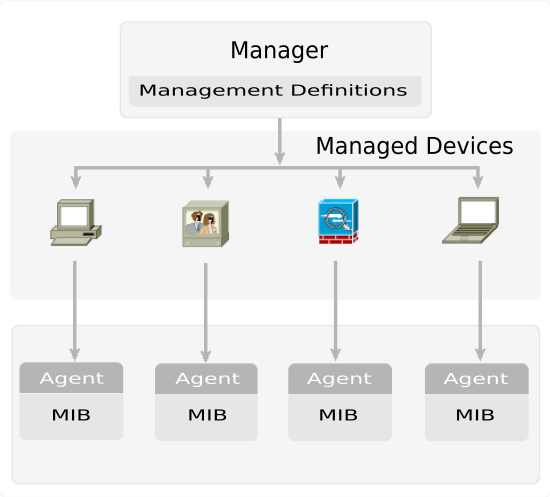

Figure 3.1: SNMP architecture

3.2.4 Client-Server Model

Client-server model is a distributed architecture design where tasks or workloads are partitioned between providers of a resource or service (server) and their requesters (client). Data storage concerns and user interface concerns are divided on this architecture.

3.2.5 Service Oriented Architecture

This kind of architecture is a form of distributed system architecture. In this context, a service is a logical view of system components defined in terms of their functionality, typically carrying out a business-level operation. It is formally defined in terms of messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. Their internal structures are abstracted away providing the benefit to incorporate any software component or application that can be "wrapped" in message handling code that allows it to adhere to the formal service definition. A service is described by machine processable meta data and only those details are exposed and have importance to the public. This description should document the services semantics. Usually systems designed using SOA are platform neutral, which means that messages are sent in a standardized format delivered trough the interfaces. Services under this architecture tends to use small number of operations with relatively large an complex messages and to be oriented toward use over a network, though this is not an absolute requirement. trough

3.2.6 Simple Object Access Protocol

Simple Object Access Protocol (SOAP) "is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment." (HADLEY et al., 2007). Simplicity and extensibility are the main design goals of this protocol.

3.2.7 Resource Oriented Architecture

"Resource Oriented Architecture (ROA) defines a specific set of web-based system design principles derived from the implementation of the REST architecture." (DUAN

et al., 2012) This architecture is a loosely coupled approach to client-server model, using a URI to directly access the resources of the devices (MAROTTA M. A., 2014).

3.2.8 Representational State Transfer

> "[Representational State Transfer (ReST)] is a coordinated set of architectural constraints that attempts to minimize latency and network communication, while at the same time maximizing the independence and scalability of component implementations." (FIELDING; TAYLOR, 2002)

A rough explanation of the main ones used to derive this style is given:

- **Client-server**: separated roles on this schema allows the components to evolve independently, improving portability of the user interface across multiple platforms and also increases scalability, simplifying the server components.

- **Stateless**: each request from the client to the server must contain all the needed information to understand the request. This constrain restricts the possibility to store context on the server. The state of the transactions are completely kept on the client side, improving visibility, reliability, and scalability. On the other hand, it may decrease network performance and reduce server's control over consistent application behaviour. (FIELDING, 2000)

- **Cache**: data within a response to a request should receive, in an implicit or explicit way, a label saying if it is cacheable or non-cacheable. When the response is cacheable, the right to reuse that response data for later and may partially or completely eliminate some interactions is given to the client, improving efficiency, scalability, and user-perceived performance. However it can decrease the reliability if the stored cache data differs from the ones obtained when requested directly to the server.

- **Uniform Interface**: all the components must communicate using a uniform interface. This way, the overall system architecture is simplified and the visibility of interactions is improved, decreasing system's coupling. Although it allows independent evolution of each component, uniform interface degrades efficiency because the data is standardized and is necessary to process it, adjusting the format to the specific application's needs.

- **Layered System**: the main goal of this constrain is to promote substrate independence. The layered system trade off is the overhead addition and latency to data processing, reducing user-perceived performance (CLARK; TENNENHOUSE, 1990), but it can be offset by caches.

- **Code on demand**: permits client extensibility, allowing codes to be downloaded and executed as applets or scripts after deployment, (FUGGETTA; PICCO; VIGNA, 1998) however it reduces system visibility, and thus is only an optional constraint within ReST.

3.2.8.1 Architectural Elements

ReST defines three different architectural elements: data elements, connectors and components. An overview of each one of them is given.

- **Data Elements**: on distributed object style, all data is encapsulated and hidden by the processing components. On ReST architectural style, information is moved from the location where it is stored to the location where it will be used by, which means that the data is moved to the processor, instead of to move the "processing agent" (e.g., mobile code, stored procedure, search expression, etc.) to the data (FIELDING; TAYLOR, 2002). It provides a hybrid of the three options that a distributed hypermedia architect has, focusing on a shared understanding of data types with meta-data and limiting the scope of what is revealed to a standardized interface. Resource, resource identifier, representation, representation meta-data, resource meta-data, and control data comprise ReST's data elements. The table 3.1 shows a relation between those elements and the modern web as is known.

Data Element	**Modern Web Examples**
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation meta-data	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Table 3.1: ReST Data Elements (FIELDING; TAYLOR, 2002)

- **Connectors**: used to encapsulate the activities of accessing resources and transferring resource representations, presenting an abstract interface for component communication. They simplify the information exchange process and enables substitutability because of the generality of the interface. As already was mentioned, all ReST interactions are stateless, so each request contains all of the information necessary for a connector to understand the request, independent of any requests that may have preceded it (FIELDING; TAYLOR, 2002). It leads to reduced consumption of physical resources and improvement of system's scalability, and it enables the analysis of isolated requests, allowing to rearrange the services dynamically. There are four different types of connectors: client, server, cache, resolver and tunnel. Table 3.1 relates each one to the modern web connectors.

Connector	**Modern Web Examples**
client	libwww, libwww-perl
server	libwww, Apache API, NSAPI
cache	browser cache, Akamai cache network
resolver	bind (DNS lookup library)
tunnel	SOCKS, SSL after HTTP CONNECT

Table 3.2: ReST Connectors (FIELDING; TAYLOR, 2002)

- **Components**: are defined according to their roles in an overall application action, and there are four different types: origin server, gateway, proxy and user agent.

> "Origin server uses a server connector to govern the name space for a requested resource. [...] A user agent uses a client connector to initiate a request and becomes the ultimate recipient of the response. [...] Intermediary components act as both a client and a server in order to forward, with possible translation, requests and responses. A proxy component is an intermediary selected by a client to provide interface encapsulation of other services, data translation, performance enhancement, or security protection. A gateway (a.k.a., reverse proxy) component is an intermediary imposed by the network or origin server to provide an interface encapsulation of other services, for data translation, performance enhancement, or security enforcement. Note that the difference between a proxy and a gateway is that a client determines when it will use a proxy." (FIELDING; TAYLOR, 2002)

Table 3.3 shows a comparison between ReST components and the modern web.

Component	**Modern Web Examples**
origin server	Apache httpd, Microsoft IIS
gateway	Squid, CGI, Reverse Proxy
proxy	CERN Proxy, Netscape Proxy, Gauntlet
user agent	Netscape Navigator, Lynx, MOMspider

Table 3.3: ReST Components (FIELDING; TAYLOR, 2002)

4 DESIGN

A "Web Interface for Accessing Option Pricer Implementations Based on the Heston Model" (TEIXEIRA; SCHRYVER; WIVES, 2013) was developed based on client-server model. The mentioned work implements Remote Procedure Calls (RPC) to start the simulations on the target-devices. It uses different protocols to connect the system's components, some of them are proprietary (GWT-RPC, for instance) and it might be a restriction, if the technology of infrastructure's components must change in a near future. A central platform is needed to measure the performance (energy consumption and execution time) of the hardware acceleration finance algorithms. It is possible then to make a parallel between network management tool and this benchmark tool. Solutions of how to make use of currently wide spread network management protocols were investigated, aiming to use standard protocols to exchange data trough the architecture's elements, avoiding compatibility problems when a new device joins the topology. The Simple Network Management Protocol (SNMP) is currently the most used protocol for network management solutions. Most devices support natively SNMP either version 1, 2 or 3. It is not difficult to implement this protocol on the ones that are not supporting it natively. Which means that, if the benchmark tool uses SNMP instead of a customized protocol, less effort is demanded during implementation, integration, support and maintenance phases. In order to achieve this goal, the first idea that stands out is to implement a gateway protocol to translate the XML data from the web interface to SNMP for the back end to process it.

4.1 Protocol Gateway

Web services are newer than the SNMP protocol on the network management field. Many research have been performed to measure whether they are more efficient or not, for example (GRANVILLE et al., 2009), (NEISSE et al., 2004), (Sloten; Pras; Sinderen, 2004) and (PAVLOU et al., 2004). The main focus of those works is to compare bandwidth consumption, response time, memory consumption and processing time. Three different translation methods were analysed: Document Object Model-based, HTTP-based and SOAP-based translations.

4.1.1 DOM-Based Translation

DOM-Based Translation infrastructure must have a component that handles the manager requests and translates them from XML to SNMP and the other way around. Therefore, the target devices can communicate with the manager with a standard

protocol.The manager has no need to understand SNMP. It adds an abstraction layer on the top of communication level between the existing components. If a DOM-based translation would be implemented on the existing infrastructure (figure 2.1), the XML-based manager will correspond to the union of web-back end servlet with the client socket and the server socket, and the gateway will be the libHeuston library. Figure 4.1 illustrates the data flow through this infrastructure.

Figure 4.1: DOM-Based Translation

4.1.2 HTTP-based Translation

HTTP-based translation is a method which also uses a gateway to translate from XML to SNMP. The difference between the DOM-based translation is that, instead of using a DOM Interface that receives and handles raw XML, the workload to translate is shared between XML-Manager and the gateway itself. The gateway receives Xpath or Xquery to translate into SNMP requests. On the existing architecture, the relation between the elements of this translation is identical to the DOM-based translation, as the main difference is on the communication between the manager and the gateway. On the benchmark tool, this method is recommended since the translations are the main working load, so if it could be shared, probably the performance will be better.

Figure 4.2: HTTP-Based Translation

4.1.3 SOAP-Based Translation

The Simple Object Access Protocol (SOAP) defines a standard method to transfer XML-encoded messages over HTTP. In other words:

> "it provides a better solution than a proprietary XML/HTTP because it is an open standard with a growing body of developers and vendors supporting it. SOAP defines a standard vocabulary and a structure for messages between communication peers, which eliminates an overhead of parsing and processing messages by a proprietary method in each gateway." (OH et al., 2002)

Many devices comes natively with SOAP support. On this case not only the link between gateway and target devices will be using a standardized protocol, likewise the communication between manager and gateway. The used protocols for a possible solution are shown in figure 4.3. This approach maps SNMP messages directly to SOAP operations.

Figure 4.3: SOAP-Based Translation

The main advantage of this solution is that there is no need of maintenance procedure on the gateway side to implement new operations. On the other hand, it is not easy for the manager to handle the requests and responses. It needs effort to implement new operations and all of them are exposed in a way that the manager should be aware of all the small details of SNMP information, increasing its complexity.

4.2 Web-Services Based Management

4.2.1 SOAP-Based Web Services

In the study "On the Management Performance of Networked Environments Using Web Services Technologies" (GRANVILLE et al., 2009) SOAP-based web services gateways are proposed and analysed. They present three different approaches: protocol-level gateway, object-level gateway and service-level gateway. It was already mentioned in last section the protocol-based gateway details. In this subsection is presented a brief explanation of the other two approaches, analysing their advantages and disadvantages.

Object-level Gateways map management information to a web service operation. For example, just one request is sent to get the name of the available scripts on determined device. When using SNMP, it requires many requests and responses. For example, suppose that one of the target devices has four scripts available. Then, when SNMP is used directly, five requests are needed: four to retrieve the script names and the last one will request another name, and will get a response that there are no more scripts available. It also receives the same number of responses. This is an improvement comparing to the protocol gateway, because the web service does not deal any more directly with the OIDs, so the gateway controls the interaction with the SNMP agent. One disadvantage may be the need to update it as soon as new information is available on the target devices MIB that must be very well defined.

Service-level gateways are those that map the management services of a MIB module. Usually it is difficult to define the services exposed by this module, as there is no concrete element that formally defines it. One operation mapped could be the association of the download request, checking and execution of a management script operation on the object-level, for example. This implementation hides SNMP details as well as all the steps to get a meaningful response, but it also implies that for each new MIB module, one new service must be implemented.

On their results was shown that usually a service-level gateway consumes less bandwidth as SNMP pure management system, once it aggregates SNMP information into a single SOAP request-reply interaction. Also, the perceived execution time is as good as the approach without web services and they are really easy to use because service details are hidden from the manager. On the other hand, they are difficult to maintain and require human interpretation of MIB services.

4.2.2 ReST-Based Web Services

As mentioned, ReST provides a loosely coupled approach to client-server model. This proposal aims to maximize the independence and scalability of the architecture components, and also to minimize latency and network communication. Those characteristics attend the requirements which the other approaches could not fulfil. Each resource has a URI as interface and HTTP is the supported protocol on transport layer. Also, instead of using web services description language (WSDL) to describe the web services, it uses Web Application Description Language (WADL). WADL is a machine-readable XML description of HTTP-based web applications, typically for ReST Web Services (HADLE, 2009). Data types supported on this approach are directly supported, without need to parse. The used standards are the web standards, for instance URL, HTTP methods, XML and mime types.

In the literature exist many studies comparing service oriented architecture and resource oriented architecture solutions for mobile hosts, mainly estimating SOAP and ReST approaches. The architectural decision were based on their results, since the target devices of this work have many similarities related to processing and network constrains. ReST proved to be more suitable on mobile hosts. The parameters used to achieve that result are very important to this research, since those are the main concern.

Al Shahwan and Moessner (ALSHAHWAN; MOESSNER, 2010)(ALSHAHWAN; MOESSNER; CARREZ, 2010) observed that the average processing time increases when the request message size increases. On their first analysed scenario, the ReST-based web-services had better performance than SOAP-based framework, mainly because SOAP demands more time to process bigger messages, while ReST has explicitly the information on the HTTP request. The second scenario they analysed scalability and reliability. They stressed with concurrent requests two mobile hosts in order to check when they will stop to respond. ReST solution shown to be more rigid and robust to changes in the number of concurrent requests because it supports cache and demands light power processing. It has a bigger threshold to start reject requests than SOAP. The SOAP-based framework starts to queue the requests to process them, starting to reject requests sooner because it needs more time to parse. Based on those results, they conclude that the ReST-based mobile web services framework is more scalable and reliable than the SOAP-based mobile web services framework. Finally they tested resource consumption and discovered that SOAP-based framework consumes a much lager amount of memory than ReST-based framework for the same message size.

Marotta, Santanna, Granville and Tarouco on their work noted that resource oriented architecture(ROA) on network management scenarios could have a response time seven times faster than service oriented architectures. Also, "[ROA] consumes 32% less power to process a query, in addition of being less influenced by other parameters"(MAROTTA M. A., 2014).

4.3 Architectural Decisions

Based on the mentioned performance results of previous work, a ReST framework was chosen to be used. At this maturity stage of the project, flexibility and scalability are needed in order to match the current requirements. In the figure 4.4 it is possible to see how currently the system is architected. The relation between the components

compared with the previous demonstrator is as following: dispatcher as the union of web back-end servlet and client socket; and working node as server socket and C++ libHeuston together. The user interface is connected to dispatcher and is completely independent and loosely decoupled from the functional part of the system.

Figure 4.4: Current Architecture - ReST-based web service

5 IMPLEMENTATION

Once the architecture type to use have been decided, the search for frameworks which better attend the new requirements was started. The system should be capable to interconnect heterogeneous hardware, with as low impact as possible, using a standard protocol to avoid compatibility problems. It has to be able to implement and integrate new benchmark sets in a effortless way. The overall scalability should be increased, without have processing or network overhead.

The library to perform the benchmark changed from C++ (libHeston) to a Python implementation (pyheston), while the analysis of the framework possibilities have been done. The project members were more familiar with Python than Java. Considering the team expertise, the system will be easy to maintain if a switch from Java to Python happens.

Frameworks and integrated development environments (IDEs) that implement ReST natively were searched. Some alternatives arrived from this: Netbeans - RESTful web services, MyEclipse - ReST web services, CakePHP, Yii framework, web2py. The use of the web2py web framework was decided to implement the new architectural style of the benchmark tool because of its useful features, as well as good documentation and support. On next section a more detailed view over this framework is presented.

5.1 Framework web2py

web2py(PIERRO, 2013) is an Open Source Web Framework distributed under the GNU Lesser General Public License version 3. It was created by a cooperation between professional community and university professors in Computer Science and Software Engineering. Massimo Di Pierro started this project and he aims three mains goals: easy of use, rapid development and security. The framework is completely backward compatible, since their first version and they are planning to not break it on future versions. Easy to run, web2py contains all the components needed to develop fully functional web applications, and the only requirement is a platform able to execute Python 2.5/2.6/2.7, or Java with Jython.

This web framework uses Model View Controller (MVC) pattern, alike the previous work (TEIXEIRA; SCHRYVER; WIVES, 2013), and provides specific tools to the developer in each one of this parts for either test, develop or design. It implements server-side form validation and postbacks which together with MVC pattern enforce good software engineering practice. Database Abstraction Layer (DAL) is used to communicate with the database. This feature permits to change the database technology transparently, without make big changes on the application.

Security is very important for information technology, either on development or infrastructure. web2py follows well established practices to handle many issues which can lead to security vulnerabilities, for example validate all inputs, escapes all outputs, renames uploaded files. This decreases the chances of a developer introducing vulnerability on the system.

Other useful features are the ticket system (from administrative point of view), portable cron, scheduler, internalization support, multiple authentication methods, role based access control and SSL support. Natively supports multiple caching methods for scalability. The community support is very helpful and web2py has an excellent documentation in many languages. It is inspired by Ruby on Rails and Django. "web2py is less verbose than Java-based frameworks and its syntax is much cleaner than PHP-based frameworks. This makes applications simpler to develop, and easier to read and maintain." (PIERRO, 2013)

5.2 MVC schema

As mentioned on design decisions chapter, the suggested benchmark tool is developed using MVC-model, enforcing the good software engineering practices. On the next subsections a more detailed view of each one of the them is presented.

5.2.1 Model

The model is composed of three different files: `db.py`, `db_model.py` and `menu.py`. The first file refers to general configuration, as the database connection and authentication configuration, for example. The database schema is defined on `db_model.py` and all the menu options and configurations as well as title and subtitle are stored on `menu.py`.

The result of the database connection using the DAL return a object, because on web2py everything is an object. Listing 5.1 shows how it is done. `db` is the connection result and all the methods, that may be applied to the database, should be done on this object.

```
1 db = DAL('mysql://username:password@host/FinanceBenchmark',pool_size=1,
    check_reserved=['all'])
```

Listing 5.1: Example of database conection

On listing 5.2 is shown how table and its fields are defined on `db_model.py`. The table `market_parameters` is build from line one to eight. The method which creates a table is `db.define_table()` and it receives as parameters the table name, followed by the fields definition. To define a field, it is used the function `Field()` and its obligatory parameters are field name followed by the field type or reference to another table field (foreign key). The DAL allows to implement constrains when defining a field, but was decided to first create all the tables and at the end implement the constrains, to make clearer and because some fields depend on the content of other fields. The requirements of `market_parameters` fields are implemented between lines ten to sixteen. For example on line ten of the given table definition, `db` is the object that represents the database used on web2py interface. It is followed by the name of the table, `maket_parameter`, and the name of the field, `correlation`. The `requires` after the name of the field is a method which is used to validate the content of a insert, update or delete operation over this specific field. It has a big set

of available functions to execute this validation. If there are more requirements over one field, it is necessary to specify into a list, otherwise just the first one will be used. Correlation on Heuston Model must have a float value between −1 and 1. Teixeira et al. (TEIXEIRA; SCHRYVER; WIVES, 2013) provides on his work a detailed explanation of each one of the parameters needed to perform the benchmarks.

```
db.define_table('market_parameters',
  Field('correlation', 'double'),
  Field('long_run_variance', 'double'),
  Field('speed_of_revertion', 'double'),
  Field('volatility_of_volatility', 'double'),
  Field('spot_price', 'double'),
  Field('spot_volatility', 'double'),
  Field('riskless_interest_rate', 'double'))

db.market_parameters.correlation.requires = [IS_NOT_EMPTY(),
    IS_FLOAT_IN_RANGE(-1,1)]
db.market_parameters.long_run_variance.requires = [IS_NOT_EMPTY(),
    IS_FLOAT_IN_RANGE(-1,1)]
db.market_parameters.speed_of_revertion.requires = [IS_NOT_EMPTY(),
    IS_EXPR('float(value)>0')]
db.market_parameters.volatility_of_volatility.requires = [IS_NOT_EMPTY
    (),IS_EXPR('float(value)>0')]
db.market_parameters.spot_price.requires = [IS_NOT_EMPTY(),IS_EXPR('
    float(value)>0')]
db.market_parameters.spot_volatility.requires = [IS_NOT_EMPTY(),IS_EXPR
    ('float(value)>0')]
db.market_parameters.riskless_interest_rate.requires = [IS_NOT_EMPTY(),
    IS_EXPR('float(value)>0')]
```

Listing 5.2: Example of table definition

All the web2py libraries are exposed to the user applications as global objects. The class `gluon.storage.Storage` extends the Python `dict` class. There are some objects which are instances of this class. `request`, `response` and `session` are the more important on the suggested implementation.

```
active_path = request.controller
response.menu = [
  (T('Home'), (request.function=='index'), URL('default', 'index'), [])
    ,
  (T('Simulation'), (request.function=='add_simulation_from_benchmark')
    , URL('default','add_simulation_from_benchmark'), []),
  (T('Results'), (request.function=='results'), URL('default','results'
    ), []),
  (T('Administrative'), (request.function=='admin'), URL('default','
    admin'),[])
  ]

DEVELOPMENT_MENU = False
```

Listing 5.3: Menu definition

The menu is constructed by the MENU helper. On listing 5.3 is shown how the `response.menu` optional parameter to pass a navigation menu tree to the view was used. It is a list of lists or of tuples. The first argument of each list/tuple is the text to be displayed for this element. The second one is a boolean to define if this item is active or not. When set to True, the MENU helper will add a "web2py-menu-active"

class to the <li> for that item, making possible to customize easily the css. The third one is a HTML helper (which could include nested helpers), and the MENU helper will simply render that helper rather than creating its own <a> tag on the respective view. The fourth argument is optional and is related to the possibles nested sub-menus. It is possible to enable a menu to help when is developing a web application by just setting `DEVELOPMENT_MENU` option to `True`. It is going to add to the menu an item called web2py with a sub-menu with links to many documentation reference and to the administrative interface of the application.

5.2.2 Controller

On controller are located the files responsible for apply updates or changes to the model. It is possible to have as many controllers as is wanted, with no restrictions. Two separated controllers are being used on this benchmark tool. One to handle the administrative interface of the server called `appadmin.py`, which is responsible for the ticket system, log, and server security polices for example, and all the functions related to the benchmark tool are defined on the `default.py`, as well as the ReST URIs and operations.

5.2.3 ReST Application Programming Interface (API)

The web2py web framework has an API which implements ReST, as was already mentioned. According to ReST architecture, each resource has its own URI. On this context it is specified as a URL. There are four methods to interact with a resource which should be implemented: create, read, update and delete. The set of those operations is known as CRUD. The system interaction is over HTTP protocol and makes use of them to pass instructions to the resources. The settled operations are POST, GET, PUT and DELETE, respectively. To understand the application request, it is necessary to parse the URL to get the parameters to the operations. web2py provides the decorator `request.restful()` which should be placed before the ReST action on the controller. This decorator allows to filter the request arguments and then process them individually.

It is necessary to associate each resource to an URL aiming to expose and to make it visible. There is a function called `parse_as_rest` which, as the name suggest, allows to parse the URL as ReST requests. When is set to 'auto', it generates all the possible combinations of resources into the system. Since this is an experimental function and is not wanted to expose all the elements of the infrastructure as a global resources, it was decided to generate all the paths to address the available resources. This was possible because there is a small set of operations to perform in order to get the required system responses and requests to perform the benchmarks.

To be able to read the ReST response also as JSON, this library had to be imported. The system uses HTTP authentication and to enable it the `HTTPBasicAuth` was imported from `requests.auth`. This is shown on listing 5.4 and those imports are placed on beginning of the default controller.

```
1 import json
2 import requests
3 from requests.auth import HTTPBasicAuth
```

Listing 5.4: Importing Libraries

Listing 5.5 shows how the ReST interface is build. Right after the beginning of

the controller definition, a decorator is placed to say that this controller should uses the ReST API. The controller function started to be defined and it was named as `api`. The third line is to enable different kind of return formats. If at the end of the ReST request an XML extension was placed, the system is going to give a response on XML format. The same happen to other formats as JSON, RSS and CSV. Inside api(), the methods GET, POST, PUT and DELETE are created. Depending on which kind of request is made, it is going to respond with a different action.

Inside the GET, the patterns to pick data from the data model are beeing defined. The patterns described here are just examples of how it is done. Those ones are exposing the `benchmark_set` resource. Line seven is making benchmark being an alias to the benchmark_set table, so now it is possible to access this resource as `http://application_host/default/api/scheduler_worker`, where "application_host" is the address to where the application is running, followed by controller name (default), desired function to access the controller and resource name. When a GET method is applied over this URI resource, it retrieves all the registers of the table benchmark_set. The next line exposes the resource in a smaller granularity, being possible now to request just a field of the registers, and not all the fields. To execute more complex queries, it is possible to use some methods, as "startswith" which returns all the exposed resources of that specific URI starting with the requested string. Example: if is wanted to retrieve all the benchmark sets starting with "asian", it is needed to send a GET to the URI `http://application_host/default/api/benchmark/asi.csv` and then an CSV view with the query is get as a result. When the list of patterns is ready, it can be passed as parameter of the function `parse_as_rest` and expose the resources. The subsequent parameters 'args' and 'vars' came from the URL itself and for the payload, respectively. After parsing the request and retrieve the data, is verified if the response was 200 which means OK, the standard response for successful HTTP requests. In positive case, a dictionary is returned with the response, otherwise the HTTP error is raised.

```
1  @request.restful()
2  def api():
3    response.view = 'generic.'+request.extension
4
5    def GET(*args,**vars):
6    patterns = [
7      "/benchmark[benchmark_set]",
8      "/benchmark[benchmark_set]/:field",
9      "/benchmark/{benchmark_set.name.startswith}",
10     "/benchmark/{benchmark_set.name}/:field",
11     "/benchmark/{benchmark_set.id}/:field"
12   ]
13
14   parser = db.parse_as_rest(patterns,args,vars)
15
16   if parser.status == 200:
17     return dict(content=parser.response)
18   else:
19     raise HTTP(parser.status,parser.error)
20
21   def POST(table_name,**vars):
22     return db[table_name].validate_and_insert(**vars)
23
24   def PUT(table_name,record_id,**vars):
```

```
25      return db(db[table_name]._id==record_id).update(**vars)
26
27    def DELETE(table_name,record_id):
28      return db(db[table_name]._id==record_id).delete()
29
30    return dict(GET=GET, POST=POST, PUT=PUT, DELETE=DELETE)
```

Listing 5.5: Restfull interface

The action POST is used to insert new records on the database. All the benchmark tool data is stored on database. This request receives a table name as parameter, from the URL, and the data to be inserted is transferred trough the payload. The DELETE action is called when is wanted to remove any previously inserted record. It needs the table name and register identifier to execute. When an update is wanted, the action PUT is the proper one to request. Its parameters are table name, record identification and the update data. It is an hybrid from POST and DELETE actions. The ReST controller definition allows to use it and call it from the user interface and access the resources.

Related to the user interface, a controller is also needed. On listing 5.6 is shown how was implemented the one responsible for add a new simulation from existing benchmark. There are comments along the code to explain each important step.

```
1  @auth.requires_login()
2  def add_simulation_from_benchmark():
3    """
4    Start new simulation from the benchmark list
5    """
6    """" Get the names of all existing benchmarks """
7    search_str = "http:///application_host/default/api/benchmark_set.json
       "
8    benchmark_row = json.loads(requests.get(search_str).text);
9
10   """ Create a list with the benchmark names"""
11   benchmarks = [benchmark_row['content'][i]['name'] for i in range(len(
       benchmark_row['content']))]
12
13   """ Get the names of all active working nodes"""
14   search_str = "http:///application_host/default/api/scheduler_worker/
       active.json"
15   working_nodes_row = json.loads(requests.get(search_str).text);
16
17   """ Create a list with all active working nodes names"""
18   working_nd = [(working_nodes_row['content'][i]['id'],
       working_nodes_row['content'][i]['worker_name']) for i in range(len(
       working_nodes_row['content']))]
19
20   """ Create a form to add new simulation based on pre-determined
       benchmarks """
21   simulation_form = SQLFORM.factory(
22                     Field('benchmark_set', requires=IS_IN_SET(
       benchmarks, zero=T('-- Select benchmark set --'))),
23                     Field('start_level','integer',default=1, requires=
       IS_EXPR('int(value)>0')),
24                     Field('final_level','integer',default=16, requires=
       IS_EXPR('int(value)>0')),
25                     Field('reference_price','double',requires=IS_EXPR('
       float(value)>0')),
```

```
                    Field('price_precision','double',requires=IS_EXPR('
    float(value)>0')),
                    Field('available_resources', 'boolean', requires=
    IS_IN_SET(working_nd, multiple=True), widget=SQLFORM.widgets.
    checkboxes.widget))

  """ Validate the input data from the form """
  if simulation_form.validate():

    """ Get all the fields of benchmark_set table which id corresponds
    to the selected one from UI  """
    search_str = "http://application_host/default/api/benchmark_set/"+
    simulation_form.vars.benchmark_set+".json"
    sel_benchmark_row = json.loads(requests.get(search_str).text);

    """ Generates the payload to insert a new job register to the
    database """
    payload = {'market_parameters': benchmark_row['content'][0]['
    mkt_param'] , 'option_parameters': benchmark_row['content'][0]['
    opt_param'], 'username': auth.user_id}
    new_job = requests.post("http://application_host/default/api/job.
    json", data=payload)

    """ if it was successfuly added, then add new algorithm parameter
    register """
    if (new_job == 200):
      payload = {'price_precision': simulation_form.vars.
    price_precision, 'reference_price': simulation_form.vars.
    reference_price, 'start_level': simulation_form.vars.start_level, '
    final_level': simulation_form.vars.final_level}
      new_alg_param = requests.post("http://application_host/default/
    api/algorithm_parameters.json", data=payload)
      if (new_job == 200):
        """ if it was successfuly added, then assign all the
    simulations to the selected working nodes """
        for resource in range(len(simulation_form.vars.
    available_resources)):
          payload = {'compute_server': resource, 'alg_param':
    new_alg_param.id, 'job_id': new_job.id}
          new_sim_param = requests.post("http://application_host/
    default/api/algorithm_parameters.json", data=payload)
          if (new_sim !=200):
            """If there are any error, tell the user """
            response.flash = T('The simulation could not begin!')
      else:
        response.flash = T('The parameters could not be added!')
    else:
      response.flash = T('The task could not be started!')
  elif simulation_form.errors:
      response.flash = T('Please check for errors on form!')
  else:
    response.flash = T('Please fill the form')
  return dict(simulation_form=simulation_form)
```

Listing 5.6: Add new simulation from existing benchmark

5.2.4 View

The files inside the section view are responsible for rendering the user interface, in this case, the web pages. There are some formats pre defined and they are known as generics. This is what allows to receive the responses in many different formats, as JSON, for example. The definition is given inside each file called generic, followed by the respective extension. Listing 5.7 exhibits the content of the file `generic.json`.

```
{{from gluon.serializers import json}}{{=XML(json(response._vars))}}
```

Listing 5.7: Structure of the file `generic.json`

The file `layout.html` contains the basic structure of the application design. On this file is defined where all elements are going to be displayed on screen and to each css class they belong. All the other pages of the benchmark tool application should extend this file. It also includes the file `web2py_ajax.html`, permitting to generate Ajax requests. Listing 5.8 shows the code used to generate the final result presented on picture 5.1.

```
{{extend 'layout.html'}}
{{=simulation_form}}
```

Listing 5.8: Structure of the file `add_simulation_from_benchmark.html`

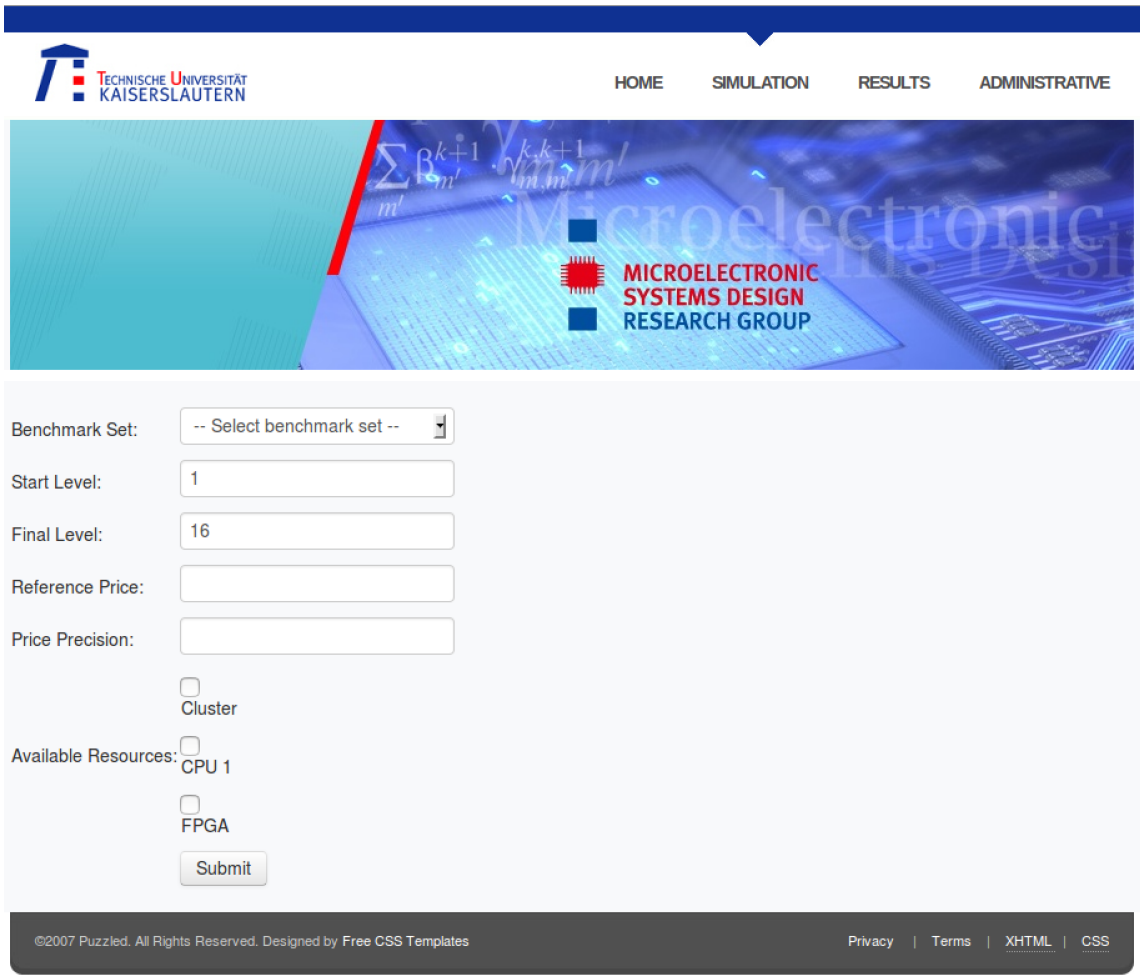

Figure 5.1: Simulation form layout

All the string which are being displayed to the user are inside the helper `T()`. This enables to translate the system in an easy way. There are some files inside the group languages of the framework which contains a dictionary to translate automatically the interface according to the user preferences.

5.3 Task Schedule

In order to perform on a working node the benchmark, the schedule module of web2py needs to be enable. The database contains the central information to schedule the tasks. On the file `db_model.py` were placed the lines described on listing 5.9. This adds three more tables on the database model: `scheduler_worker`, `scheduler_task` and `scheduler_run`. All the information about tasks running, available resources, last heartbeat, run_result, are stored at this tables.

```
from gluon.scheduler import Scheduler
scheduler = Scheduler(db)
```

Listing 5.9: Enable scheduler

Despite enabling the scheduler module, it is needed to define the task format. A new task on this context is a new simulation. On listing5.10 is shown how one of the benchmarks is triggered and the results stored on database. Defining tasks and implementing them in such a modular way allows us to easily change the current scheduler engine to another more dynamic.

```
def new_sim(args):
    import subprocess
    """ begin task execution """
    output = subprocess.check_output(['python3','/var/lib/pyheston/
    heston_web2py.py', args])
    results=output[1:-2].split(',')
  """ Stores results on database """
    payload = {'energy':results[0],'runtime_value':results[1],'price':
    results[2],'precision_value':results[3]}
    r = requests.post("http://127.0.0.1:8000/webpricer/default/api/
    results.json", data=payload)

    return results
```

Listing 5.10: New task definition

6 CONCLUSION

The present investigation has compared different ways to interconnect and execute algorithms in distributed heterogeneous hardware. The development of a new benchmark tool was undertaken to design a more scalable, flexible and standard solution, compared to the previous work. Returning to the proposed goals at the beginning of this study, it is now possible to state that the change of the architectural style was the main factor to achieve better results.

At the beginning of this work many issues related to the fast change of the requirements as well as changes on other infrastructure components had happened. Flexibility was a very important aim to achieve on the context of this research. The first approach was to reuse and remodel the current version of the benchmark tool, aiming to save implementation time. After the first tries to modify the existing tool, it was realized that there were many changes and it will continue changing. This was the main reason to rethink on the architectural model of the tool.

The second important decision was the framework. Some of them were tried for a while and it was noticed that many of the tries had not desirable restrictions. Many claims to easily implement ReST applications, but this fact was not true if you were not familiar with the framework or the library. This was one of the advantages of using web2py, because the deployment and development time were reduced. The good documentation and the helpful community support are differential comparing to others tools.

These research findings suggest that in general resource oriented architecture with representational state transfer interfaces is the best choice for network management of target-devices with memory and processing constrains. Employing this architectural style, the system components coupling was reduced, favouring the scalability improvement. The chosen framework contributed to keep the good software engineering practices, facilitated the implementation of basic security techniques and other relevant features, as internalization support. The methods used to perform benchmarks may be applied to execute other algorithms on distributed heterogeneous hardware. If the scheduler is changed to another, it may be possible to execute smarter decisions, considering queue and time to processing based on benchmark results, for example.

The current investigation was limited to a small controlled testing context. However, with a limited scope, caution must be applied, as the findings might not be transferable to bigger infrastructures. It is needed to perform a workload test to decide which is the suitable number of schedulers, as well the number of supported database concurrent connections.

Further research might explore the security concerns of this system. Despite small

issues being addressed by the framework and during the development phase, the overall security of the system is insufficient and needs special attention. A further study regarding the role of the scheduler would be worthwhile, analysing how to better schedule the tasks. The mentioned change may allow expand the system to not just perform benchmarks, but also perform distributed computation aiming better use of integrated resources.

AppendixA

A.1 Previous Database Schema

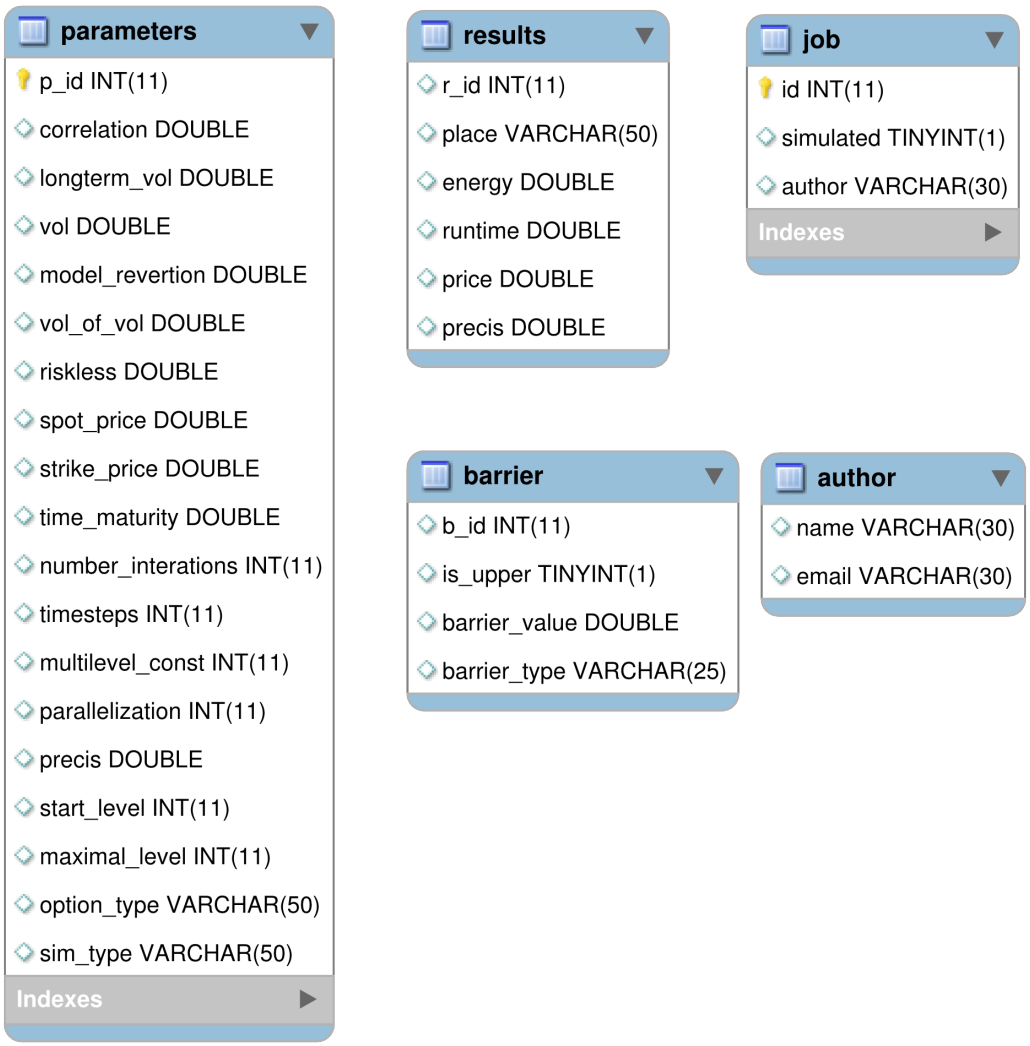

Figure A.1: Previous Database Schema

A.2 Current Database Schema

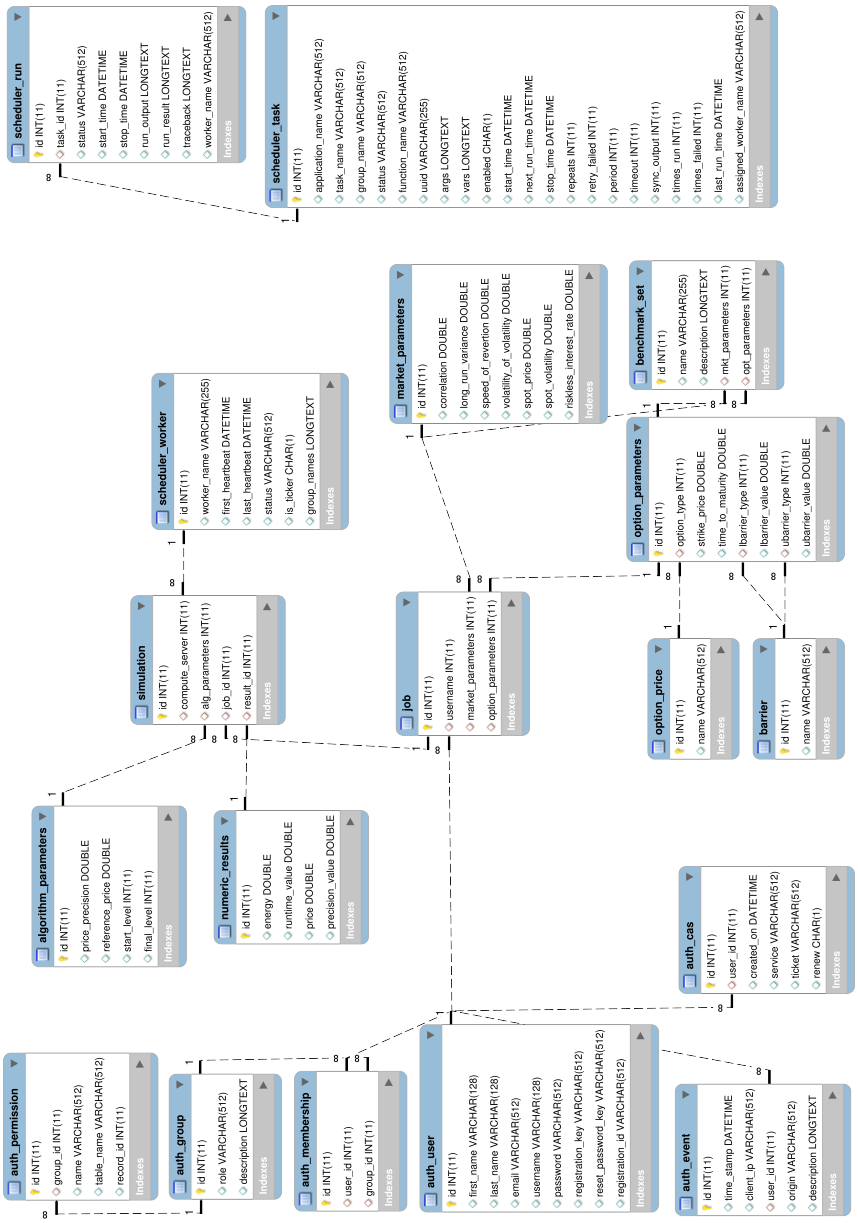

Figure A.2: Current Database Schema

REFERENCES

ALSHAHWAN, F.; MOESSNER, K. Providing SOAP Web Services and RESTful Web Services from Mobile Hosts. In: INTERNET AND WEB APPLICATIONS AND SERVICES (ICIW), 2010 FIFTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.174–179.

ALSHAHWAN, F.; MOESSNER, K.; CARREZ, F. Evaluation of Distributed SOAP and RESTful MobileWeb Services. **International Journal on Advances in Networks and Services**, [S.l.], v.3, n.3 & 4, p.447 – 461, November 2010. 2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org.

CASE, J. et al. **Simple Network Management Protocol (SNMP)**. [S.l.]: IETF, 1990. n.1157. (Request for Comments).

CLARK, D. D.; TENNENHOUSE, D. L. Architectural Considerations for a New Generation of Protocols. In: ACM SYMPOSIUM ON COMMUNICATIONS ARCHITECTURES & PROTOCOLS, New York, NY, USA. **Proceedings...** ACM, 1990. p.200–208. (SIGCOMM '90).

D. BOOTH H. HAAS, F. M. E. N. M. C. C. F. D. O. **Web Services Architecture**. Latest version available at http://www.w3.org/TR/ws-arch/ .

DUAN, K. et al. Composition of Engineering Web Services with Universal Distributed Data-flows Framework Based on ROA. In: THIRD INTERNATIONAL WORKSHOP ON RESTFUL DESIGN, New York, NY, USA. **Proceedings...** ACM, 2012. p.41–48. (WS-REST '12).

FIELDING, R. et al. **Hypertext Transfer Protocol – HTTP/1.1**. Updated by RFCs 2817, 5785, 6266, 6585, RFC 2616 (Draft Standard).

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Tese (Doutorado em Ciência da Computação) — University of California, Irvine.

FIELDING, R. T.; TAYLOR, R. N. Principled Design of the Modern Web Architecture. **ACM Trans. Internet Technol.**, New York, NY, USA, v.2, n.2, p.115–150, May 2002.

FUGGETTA, A.; PICCO, G. P.; VIGNA, G. Understanding Code Mobility. **IEEE Transactions on Software Engineering**, Los Alamitos, CA, USA, v.24, n.5, p.342–361, 1998.

GRANVILLE, L. Z. et al. **Handbook of Research on Telecommunications Planning and Management for Business**. [S.l.]: IGI Global, 2009. p.724–741.

HADLE, M. **Web Application Description Language**. [S.l.]: W3C, 2009. W3C Member Submissionn, http://www.w3.org/Submission/wadl/.

HADLEY, M. et al. **SOAP Version 1.2 Part 1**: messaging framework (second edition). [S.l.]: W3C, 2007. W3C Recommendation, http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

HARRINGTON, D.; PRESUHN, R.; WIJNEN, B. **An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks**. Updated by RFCs 5343, 5590, RFC 3411 (INTERNET STANDARD).

JIN, Q.; LUK, W.; THOMAS, D. On Comparing Financial Option Price Solvers on FPGA. In: FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES (FCCM), 2011 IEEE 19TH ANNUAL INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2011. p.89–92.

LAI, Y.; SPANIER, J. Applications of Monte Carlo Quasi-Monte Carlo methods in finance: option pricing. In: THIRD INTERNATIONAL CONFERENCE ON MONTE CARLO AND QUASI-MONTE CARLO METHODS. **Proceedings...** Springer-Verlag, 1998. p.284–295.

MAROTTA M. A., B. C. B. R. J. G. L. Z. T. L. M. R. Evaluating Management Architectures for Internet of Things Devices. In: WIRELESS DAYS (WD), 2014 IFIP, Rio de Janeiro, Brazil. **Anais...** [S.l.: s.n.], 2014. p.11–12. Submitted.

MCCLOGHRIE, K.; ROSE, M. **Management Information Base for Network Management of TCP/IP-based internets:mib-ii**. Updated by RFCs 2011, 2012, 2013, RFC 1213 (INTERNET STANDARD).

NEISSE, R. et al. Implementation and bandwidth consumption evaluation of SNMP to Web services gateways. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, 2004. NOMS 2004. IEEE/IFIP. **Anais...** [S.l.: s.n.], 2004. v.1, p.715–728 Vol.1.

OH, Y.-J. et al. Interaction Translation Methods for XML/SNMP Gateway. In: DSOM. **Anais...** Springer, 2002. p.54–65. (Lecture Notes in Computer Science, v.2506).

PAVLOU, G. et al. On management technologies and the potential of Web services. **Communications Magazine, IEEE**, [S.l.], v.42, n.7, p.58–66, July 2004.

PIERRO, M. D. **web2py (5th Edition)**. [S.l.]: Experts4Solutions, 2013.

ROSE, M.; MCCLOGHRIE, K. **Structure and identification of management information for TCP/IP-based internets**. [S.l.]: IETF, 1990. n.1155. (Request for Comments).

SANCHEZ, A.; ALVARADO-NAVA, O.; MARTINEZ, F. Network monitoring system based on an FPGA with Linux. In: TECHNOLOGIES APPLIED TO ELECTRONICS TEACHING (TAEE), 2012. **Anais...** [S.l.: s.n.], 2012. p.232–236.

SCHMERKEN, I. **Deutsche Bank Shaves Trade Latency Down to 1.25 Microseconds**. "[Online; accessed 31-March-2013]", http://hdl.handle.net/10183/86278).

SCHRYVER, C. de et al. An Energy Efficient FPGA Accelerator for Monte Carlo Option Pricing with the Heston Model. In: IEEE INTERNATIONAL CONFERENCE ON RECONFIGURABLE COMPUTING AND FPGAS (RECONFIG). **Proceedings...** [S.l.: s.n.], 2011. p.468–474.

SCHRYVER, C. de; TORRUELLA, P.; WEHN, N. A Multi-Level Monte Carlo FPGA Accelerator for Option Pricing in the Heston Model. In: IEEE CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE (DATE). **Anais...** [S.l.: s.n.], 2013.

Sloten, J. van; Pras, A.; Sinderen, M. van. On the standardisation of Web service management operations. In: OPEN EUROPEAN SUMMER SCHOOL AND IFIP WG6.3 WORKSHOP - ADVANCES IN FIXED AND MOBILE NETWORKS, EUNICE 2004, 10., Tampere, Finland. **Anais...** Tampere University of Technology, 2004. p.143–150.

STALLINGS, W. **SNMP, SNMPv2, SNMPv3, and RMON 1 and 2**. [S.l.]: Addison-Wesley, 1999.

TEIXEIRA, G. A.; SCHRYVER, C. de; WIVES, L. K. **A web interface for accessing option pricer implementations based on the Heston model**. "[Online; accessed 23-Mai-2014]", http://www.advancedtrading.com/infrastructure/229300997.

WOOD, L. et al. **Document Object Model (DOM) Level 1**. [S.l.]: W3C, 1998. W3C Recommendation, http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001.