Web Service Micro-Container for Service-based Applications in Cloud Environments

Conference Paper *in* Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE · June 2011

DOI: 10.1109/WETICE.2011.51 · Source: DBLP

4 authors:

Mohamed Mohamed
IBM Resarch Almaden
20 PUBLICATIONS **82** CITATIONS
SEE PROFILE

Sami Yangui
Concordia University Montreal
27 PUBLICATIONS **101** CITATIONS
SEE PROFILE

Samir Moalla
University of Tunis El Manar
10 PUBLICATIONS **44** CITATIONS
SEE PROFILE

Samir Tata
Institut Mines-Télécom
119 PUBLICATIONS **747** CITATIONS
SEE PROFILE

2011 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises

Web service micro-container for service-based applications in Cloud environments

Mohamed Mohamed[1], Sami Yangui[1], Samir Moalla[1] and Samir Tata[2]
[1] Faculté des Sciences de Tunis, 2092 Tunis EL Manar, Tunisia
mohamedmohamed@orange.tn, yangui.sami@yahoo.fr, samir.moalla@fst.rnu.tn
[2] Institut TELECOM, TELECOM SudParis, UMR CNRS Samovar, 91011 Evry Cedex, France
Samir.Tata@it-sudparis.eu

***Abstract*— Cloud computing describes a new supplement, consumption, and delivery model for IT services based on Internet protocols, and it typically involves provisioning of dynamically scalable and often virtualized resources. In this paper, we propose to design and implement a new service micro-container to address scalability by reducing memory consumption and response time. We propose to dedicate a services micro-container for each deployed service and thus avoid the processing limits of classical services containers. Our micro-container is evaluated and compared to conventional Web containers to highlight our contribution.**

I. Introduction

Web services can be seen as a pillar block for achieving electronic B2B transactions. More and more companies are using Web services to achieve transactions with their partners and/or offer on-line services. For instance, in a Mckinsey Quarterly survey [11] conducted in 2007 on more than 2800 companies worldwide, 80% are using or planning to use Web services. Among these companies, 78% says that the Web services technology is among the three most important technologies to their business.

To make their services online, companies can set up their own infrastructure or can adopt the new economic model offered by Cloud Computing. Cloud computing describes a new supplement, consumption, and delivery model for IT services based on Internet protocols, and it typically involves provisioning of dynamically scalable and often virtualized resources. There is a no consensus on a definition of Cloud computing relying on the definition of some twenty experts[8]. Foster et al. [10] define cloud computing as a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.

Although there is no consensus or definition of the concept of cloud but there are few common key points in these definitions. First, Cloud computing is a specialized distributed computing paradigm [10]; it differs from traditional ones on the fact (1) it is massively scalable, (2) it can be encapsulated as an abstract entity that delivers different levels of services to customers outside the Cloud, (3) it is driven by economies of scale and (4) can be dynamically configured (via virtualization or other approaches) and delivered on demand.

In this work, we aim at showing that classical service containers such as Axis2 are not adequate to be used for services management in a context of Cloud computing. Indeed classical service containers are not in line with characteristics of Cloud environments. Infact, they are not designed for elasticity. For example, the occupied memory of these classical containers is limited to the size of the memory of the physical node on which they are deployed even though virtualization techniques are used.

In this paper, we propose to design and implement a new service micro-container to make the tasks performed previously by classical service containers possible in a Cloud environment.

The micro-container that we propose should be as lightweight as possible for an optimal usage of Cloud resources and ensure good performance in terms of response time and memory consumption. Regarding the issue of scalability, we propose to dedicate a services micro-container for each deployed service. So, we will have as many micro-containers as deployed services and thus avoid the processing limits of classical services containers. The new size limit of memory consumption will be the size limit of all physical nodes of the Cloud environment. One can even push up this size limit when considering hybrid Cloud environments. So the actual limit of service deployment would be the limit of the all available physical resources in the Cloud. In addition of that, the deployment process will be very easy and summarized in enclosing a service within its own micro-container.

This paper is organized as follows. Section II presents a state of the art of Cloud computing environments and the motivations of our work. In Section III, we present conception and the architecture of our service micro-container. In Section IV, presents the implementation and the experiments of our realization. Finally, in Section V we conclude our paper and present our future work.

II. State of the art and motivation

Cloud providers offer different API to access to their Cloud services. We can cite among other the following APIs: Amazon API [12], GoGrid's API [13], Sun's Cloud API [14] and VMware's vCloud [15]. The Service Oriented Architecture (SOA) is one of the principle architectures related to cloud computing; hence we notice the increasing use of "Everything

1524-4547/11 $26.00 © 2011 IEEE
DOI 10.1109/WETICE.2011.51

as a service" terms like IaaS for infrastructure as a service, PaaS for platform as a service and SaaS for Software as a service and so on [18][16].

Taxonomy of cloud computing systems shows that all the existing systems are limited to a programming framework which makes the use of those clouds difficult, since cloud clients need to use the related programming language before using the cloud [16]. For example, Amazon imposes Amazon Machine Image (AMI) and Amazon MapReduce framework, Force.com imposes Apex language for database service, Azure imposes Microsoft .Net [19][20], Google App Engine imposes MapReduce programming framework [19], and so on.

The use of the SOA approaches in Cloud-based applications leads to the use of service containers to manage the life cycle of the provided services part of the deployed applications. After studying different architectures of service containers, we realized that all of them are not able to scale among many physical machines. Any of those containers, Axis2 included, can be deployed physically just on one machine, so the cloud using such containers will reach its limits when this machine uses its entire resources even if the other machines are charge free. We can say that the cloud's limit is the same limit of the machine in which we deployed the service container. This machine presents a bottleneck in every cloud using such containers for service-based applications.

Amazon EC2, Force.com and Eucalyptus are examples of clouds using Axis2 container to manage their deployed services. Axis2 is one of the service containers that can handle a big number of services in the same time and response to client's queries in an acceptable time [17].

We think that if every service could be deployed separately, the cloud can really scale easily till it reaches its real limit which are the limits of all the resources of the cloud. Every service can be deployed anywhere in the cloud with the minimal use of its resources. We got the idea to create a web service micro-container that is able to contain just one service. This micro-container provides the minimal functionalities to manage the life cycle of the deployed web service. We can deploy as many micro-containers as it is possible on any machine, if this machine reaches its limit we can deploy on a second one then on a third and so on. With this idea we will show that we use the minimal resources to encourage the pay-as-you-go model of cloud computing and we can enforce the elasticity of cloud because we use just the resources needed no more and no less.

III. Service micro-container

Doudoux [1] presents containers as mechanisms for managing the lifecycle of components that run in them. The container hosts and provides services that can be used by applications during their execution. To deploy an application in a container, one must mainly provide two elements: the application with its all components (compiled classes, resources, etc.) included in an archive or a module and a deployment descriptor file contained in the module that specifies the container options to run the application. For example, for the J2EE platform, there are several types of containers: Web containers, for servlets and JSP, EJB containers, for EJB, and client containers for applications on standalone terminals using J2EE components.

In line with the definition given in Wikipedia [2], we can define a Web container as an application that implements the communication contract between different application components obeying a distributed architecture. This contract specifies a runtime environment for Web components including safety and competition management, lifecycle, transactions, deployment and other services. Web containers can generally use their own Web server and also be used as a plug-in a dedicated Web server (as is the case with Apache servers or Microsoft IIS). Examples of Web containers are Tomcat (a J2EE container implementation) and Axis2 which are open sources projects from Apache.

In this paper, we shall focus on a particular type of Web containers called services containers. Of course, services containers meet all specifications for the Web container already mentioned and provides, in addition, support and management of Web services. As far as we know, Tomcat and Axis2 are both the most popular Web container used by developers.

For optimality and performance constraints, features of our micro-containers will be as minimal as possible. After studying the features provided by conventional service containers and other container architectures. We drew up a list of basic features that should satisfy our micro container. For example, we failed to incorporate a safety module for managing access since it is a prototype and management competitions module since we are assuming a single Web service per container. This list reflects directly the different components that make up our micro-container architecture and ignores the extensions and classical add-ons of conventional service containers. Obviously, this does not affect the main process of our micro container (Web services hodting, interaction with clients,etc.).

We believe that we will need the following modules:

- A transport module for user requests reception and for sending responses,
- A SOAP and XML processing module for analyzing SOAP messages elements and interpret/generate XML contents,
- A process module for the invocation and processing deployed Web services,
- A deployment module for deploying a deployed Web service in the micro-container.

The architecture of our micro-container is formed by several components detailed in Fig.1 below:

- A deployment platform for addressing the source archives and correspondent WSDL for the generation of a corresponding micro-container and service deployment,
- Micro-container to host the deployed Web service and handle various clients requests,
- Thin clients to invoke services via micro-containers

The deployment framework is responsible of the generation of the micro container component. Specifically, after analyzing the WSDL file of the service by the WSDL parser component and after receiving all its source archive (Fig. 1, Action 1), the processing module chooses the type of binding to use with the service from the generic communication module and switch

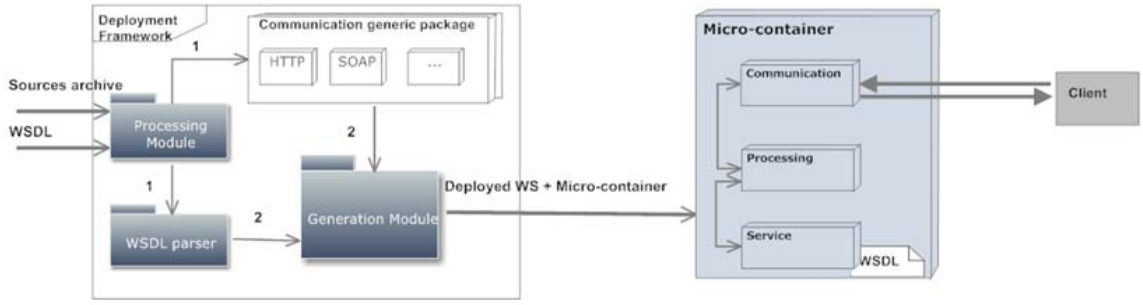

Fig. 1. System Architecture

this decision to the generation module which generates and deploys the specific micro-server with the necessary modules (Fig. 1, Action2).

This mechanism ensures that we are decreasing the memory unlike many containers engaging all the modules for a service which may use just one module. This also helps to identify the entire process of analysis, generation and deployment of the total load of the micro-container and thus ensure its optimum performance.

Micro-container component is responsible of managing the communication with the client, holding the service and processing all the messages incoming or outgoing the micro-container. It is composed of the necessary modules for the deployed service, no more, no less. The architecture of the micro-container is shown in Fig. 1 and shows three main components. Each of them ensures a feature from features list introduced above.

The micro-container is composed of 3 parts:

- Communication module to establish communication and to support connection protocols,
- Processing module to process ingoing and outgoing data the server (packing and unpacking data),
- Service module to store and invoke the requested Web service.

The next section presents the implementation of our micro-container and some experiments related to response time and memory consumption.

IV. Implementation & Experiments

In this section we will detail the implementation of our micro-container and detailed the experimentation we have done that compare the performance of our micro-container against the Axis2 container.

A. Implementation

The implementation took place in three phases. We have first developed a minimal Java deployment framework, which allows developers to deploy a Java Web service on a hard-coded micro-container before deploying both of them in the Cloud. After that, we developed the generation module by generating and deploying an optimal and minimal micro-container. We have also developed Java clients which send requests to the service micro-containers and display results returned by the deployed Web services.

At the first iteration of our implementation, we mainly focused on automating generation and deployment process of a service micro-container from a WSDL and source archives provided by a developer. Then, in the second iteration, our concern was to alleviate as much as possible the generated micro-container for performance reasons and scalability constraints. In other terms, we had to refine the generation process. To do this, we defined a generic component of communication in the deployment platform to identify and contain most all the communication protocols that can support a service. The generation process is based primarily on results Bindings components detected by the WSDL parser and secondly by the activation of corresponding communication modules from the generic communication module.

In addition, before generating the communication module of the micro-container, we have imagined the scenario that traces the sequence of events from a request reception until sending a response. Indeed, for a SOAP over HTTP communication, the execution of this module takes place in four steps: (1) receipting of the client request, (2) extracting HTTP SOAP envelopes, (3) invocation of requested Web service and (4) building of the response message and send it to the client.

The generation module aims at supporting several communication protocols between the service micro-container and the clients. Later, during the deployment, only necessary communication protocol is encapsulated in the micro-container. Fig. 2 represents UML sequence diagram which describes the service Invocation's scenario and the different interactions between the services micro-container and clients.

A client sends its request to the service micro-container via a specified port. The micro-server intercepts this request and associates it to a new connection with this client. Parameters, which represents deployed Web service arguments, are extracted from the client HTTP request and unpacks parameters where side use an instance of Message_Processor. After Web service invocation, Message_Processor packs execution results, build the message response and sends it back to the client before closing the connection. By analogy with deployment Web services procedures on conventional Web containers, the developer must provide service's sources archive and provide service's WSDL file. The developer must follow instructions

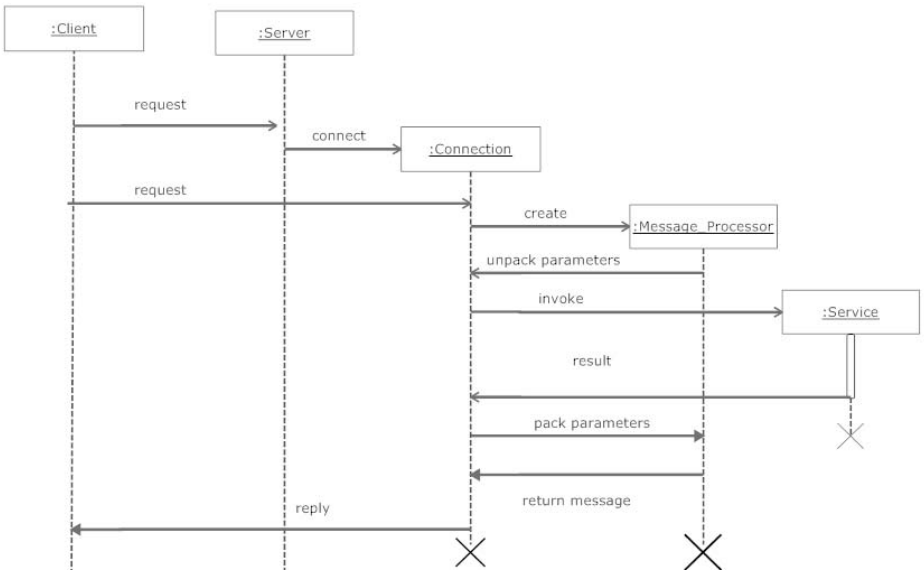

Fig. 2. Invocation scenario

of the deployment descriptor of the Web container. Similarly, our deployment process is greatly inspired by this process. Fig. 3 below represents an UML sequence diagram which describes this procedure. We remind that we deploy a single Web service on the micro-container in our case.

Each instance of the micro-container uses a communication protocol with clients. To deploy a service, the user has to provide a WSDL file and its related sources archive. The Deployer analyses the WSDL file and detects which type of micro-container will be deployed with the Web service. This generation is related to the binding specified in WSDL file. For example, if the service description denotes that the service supports a HTTP connection, the Deployer will instantiate a HTTP micro-server and the micro-container started is ready to receive any HTTP client request via the specified port of the Web service.

In the next subsection, we present the experiments we have conducted to validate our proposal.

B. Experiments

To evaluate performance of our proposed micro-container, we have considered a couple of criteria:

- Response time: Time taken by a service container between request reception instant and response sending instant.
- Memory consumption: Memory size necessary to load and process deployed service in services container after receiving a request.

To perform these tests, we chose to evaluate the performance of our services micro-container opposite to Axis2. This choice was motivated by the performance of Axis2. Indeed, as far as we know, Axis2 is one of the best efficient conventional services containers.

We have also decided to perform tests in one machine even for the micro-container. Axis2 can be deployed on one machine. Nevertheless our container can be deployed over more than one physic machine, hence, if our container defeats Axis2 using just one machine so we can confirm that it can defeat Axis2 more easily on large scales Cloud context included, because our micro-container can scale without any interaction between the services deployed unlike any other container.

The machine that we have used for experiments is a Siemens Fujitsu machine with a Pentium Dual Core 2.5 GHz, 2 Gigabyte of RAM and a Microsoft Windows XP Professional SP3 as operating system.

We have also developed a test collection generator to obtain thousands of generated Web services code archives and their WSDL files. The functionality which implements these Web services is the same: sum of two integers.

Fig. 4 below shows the different stored values for response time experiments.

Axis2 time response increases proportionally with the number of deployed services. It represents the time needed for Axis2 to load a service, update indexations and contexts and execution. The interpretation time of request and the time for the building of a response from a result are always the same. For our micro-container, the response time is almost the same for all the experiences, because every instance of the micro-container is independent from the others, hence, we can deploy as many micro-containers as it is possible regarding the available resources in the machine without affecting the response time. In this environment, Axis2 reached its limit when 5700 services are deployed. Throwever, our micro container reached more than 8000 deployed services using our defined approach with the same performances and without any problem. During these experiments, we had to make a choice between (1) test by comparing Axis 2 performance versus a single instance of the micro container performance (2) or test by comparing total CPU time between all instances of deployed micro containers

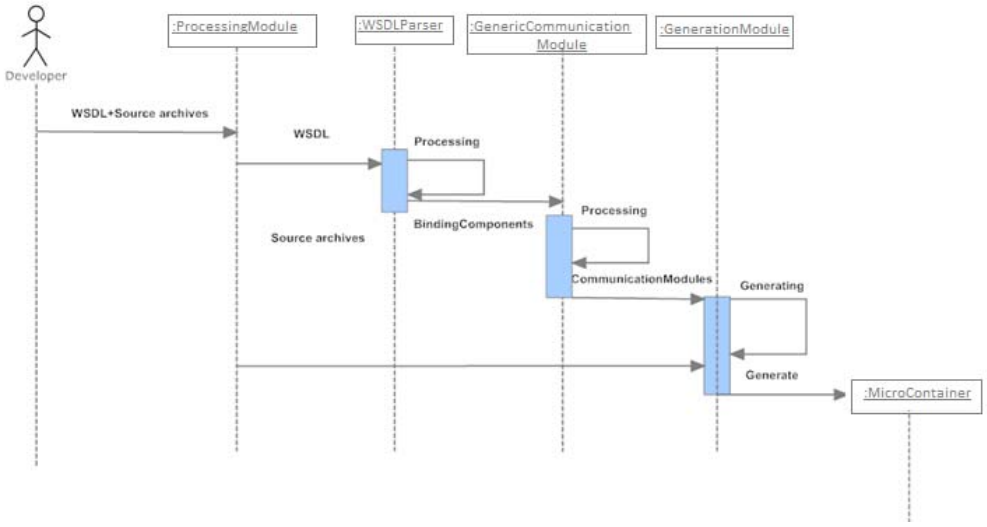

Fig. 3. Deployment scenario

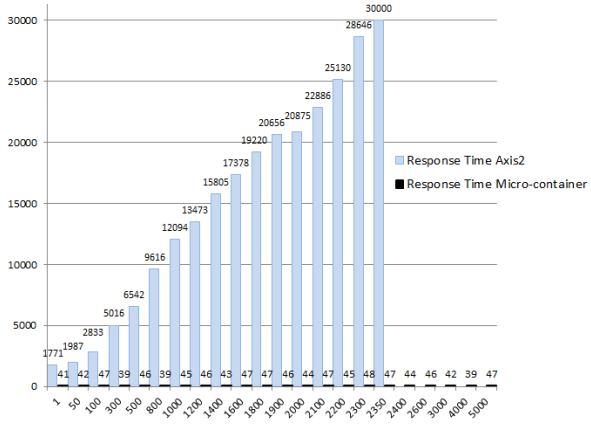

Fig. 4. Time response evolution (Axis2 vs Micro-container) -int inputs

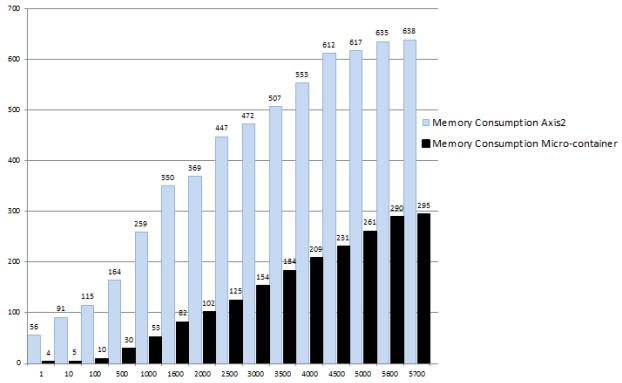

Fig. 5. Memory consumption (Axis2 vs Micro-container)

running in parallel versus Axis2. We opted for the first test plan because we chose to comprare performance of the two application servers with the same test collection (deployed services).

Fig. 5 shows the different stored values for memory consumption experiments. We have repeated the same experiments in the same environment for memory criteria. We notice that the memory usage is linear, increasing with the number of services in the two sides. Results show the savings of the micro-container against Axis2 in memory usage. That is due to the large number of files generated using Axis2 container for each web service deployed (archives, indexation, temporary files, context files...). Those files' size is larger than a micro-container's size.

For the first criterion of our experiments, in addition to the variation of the number of deployed services, we have also tried to diversify as much as possible our experiments. Specifically, each time, we have changed data types provided to services for the same test collections. As our implementation is in JAVA we decided to use JAVA primitive types (int, long, double, etc.). For experiments with objects type data, we manipulated Strings. The goal is not only to see the impact of these types changes on Axis2 response time, our micro-container response time but also the evolution of the difference in performance already registered with the integers in the initial experiments. To do so, we slightly modified the source of services and the collection tests generation program.

Fig. 6 and Fig. 7 below show the different stored values for response time experiments respectively for double inputs and String inputs.

We note that Axis2 reached much faster (limit in terms of number of deployed services) the threshold of the timeout (30000 ms) for integer type inputs. This is explained by the fact that the average processing times for String type data or double type are less than the time required for the manipulation of integers. This slight increase has also been observed on the response time of our micro-container. We can therefore conclude that the difference between response time of our

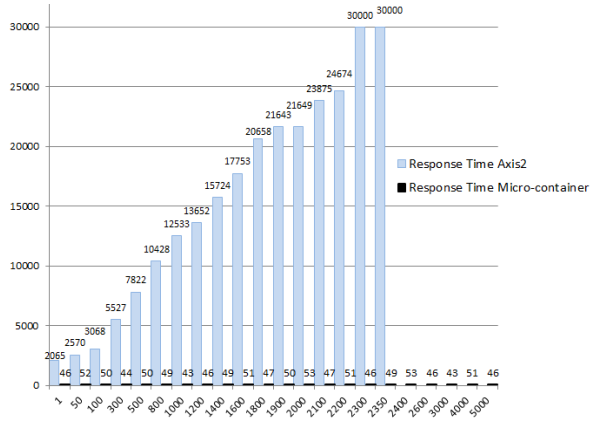

Fig. 6. Time response (Axis2 vs Micro-container) -double inputs

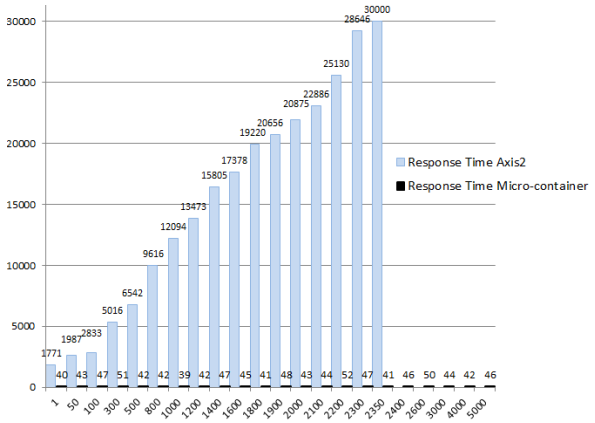

Fig. 7. Time response (Axis2 vs Micro-container) -string inputs

micro-container and Axis2 recorded during experiments on integers inputs has been accentuated in these experiments.

Finally, we tried to vary the number of inputs to deployed services at invocation to test the reaction of our micro-container facing Axis2. The choice of such experiment was motivated by declining performance Axis2 from a given number of inputs. Fig. 8 shows the different stored values for response time experiments for different numbers of integer inputs.

V. Conclusion

The work presented in this paper is based on a simple idea that consists in dedicating a micro-container to each deployed service in a context of cloud environment. Only necessary resources, such as communication protocol, are encapsulated in the micro-container to host the deployed service. We have defined the architecture of the proposed micro-container and the deployer that is in charge of generating the micro-container. We have in addition presented the implementation of our micro-container. During, the first experimentations we have done challenging Axis2 are rather very encouraging. They clearly show the gain we can have when we opt for our micro-container.

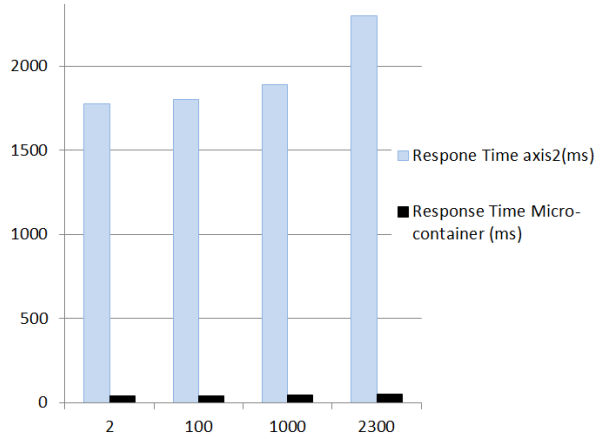

Fig. 8. Time response depending inputs number (Axis2 vs Micro-container

In our future works, we are planning to refine more our experiments. We will push further our context of experimentation and use our cloud computing infrastructure composed of 256 cores.

References

[1] http://www.jmdoudoux.fr/java/dej/chap048.htm (2010).
[2] http://en.wikipedia.org/wiki/Web_container (2010).
[3] http://searchsoa.techtarget.com/definition/Tomcat (2011).
[4] E. Langlet. Apache Tomcat 6, Guide d'installation du serveur Java EE sous Windows et Lunix Livre des éditions ENI Collection Ressources Informatiques. ISBN : 978-2-7460-4162-2, pp121-126
[5] http://www.javajazzup.com/issue11/page13.shtml (2011).
[6] S. Perera, C. Herath, J. Ekanayake, E. Chinthaka, A. Ranabahu, D. Jayasinghe, S. Weerawarana, G. Daniels. Axis2, Middleware for Next Generation Web Services. In: IEEE International Conference on Web Services, ICWS, 2006.
[7] A. Dhesiaseelan, V. Ragunathan. Web Services Container Reference Architecture (WSCRA). In the IEEE International Conference on Web Services,ICWS, 2004.
[8] Twenty Experts Define Cloud Computing, SYS-CON Media Inc, http://cloudcomputing.sys-con.com/read/612375_p.htm (2011)
[9] J. Silvestre. Economies and Diseconomies of Scale. In the New Palgrave: A Dictionary of Economics, v. 2, pp 80–84, 1987.
[10] I. Foster, Y. Zhao, I. Raicu and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In the IEEE Grid Computing Environments (GCE08), (2009).
[11] The Mckinsey Quarterly: How businesses are using web 2.0. A mckinsey global survey http://www.mckinsey.de/downloads/publikation/mck_on_bt/2007/mobt_12_How_Businesses_are_Using-Web_2_0.pdf (2011).
[12] J. Varia. Amazon white paper on cloud architectures. http://aws.typepad.com/aws/2008/07/white-paper-on.html (2008).
[13] GoGrid web site. http://www.gogrid.com (2009).
[14] The sun cloud API. http://kenai.com/projects/suncloudapis (2009).
[15] vCloud API programming guide, Tech. Rep., VMWARE Inc., 2009.
[16] B. Prasad Rimal,E. Choi,I. Lumb. A Taxonomy and Survey of Cloud Computing Systems. In the fifth International Joint Conference on INC, IMS and IDC, 2009.
[17] L.R. Merino, L.M. Vaquero, V. Gil, F. Galan, J. Fontan, R.S. Montero, I.M. Liorente. From infrastructure delivery to service management in clouds. In: Future Generation Computer Systems (26) pp 1226-1240, 2010.
[18] A. Goscinski, M. Brock. Toward dynamic and attribute based publication, discovery and selection for cloud computing. In: Future Generation Computer Systems (26) pp 947-970, 2010.
[19] Microsoft, Azure. http://www.microsoft.com/azure/default.mspx (2009).
[20] Google, App Engine. http://code.google.com/appengine/ (2009).