Efficient Web Services Selection based on QoS through a Distributed Parallel Semantic Approach

Luis H. V. Nakamura*†, Pedro F. do Prado*, Rafael M. de O. Libardi*, Luiz H. Nunes*,
Rodolfo I. Meneguette†, Julio C. Estrella*, Regina H. C. Santana*, Marcos J. Santana*, Stephan Reiff-Marganiec ‡
* *Institute of Mathematics and Computer Science (ICMC), University of São Paulo (USP), São Carlos-SP, Brazil*
Email: {nakamura, pfprado, mira, lhnunes, jcezar, rcs, mjs}@icmc.usp.br
† *Federal Institute of São Paulo (IFSP), Catanduva, São Paulo, Brazil*
Email: {nakamura, meguette}@ifsp.edu.br
‡ *University of Leicester, University Road, Leicester, LE1 7RH - UK*
Email: srm13@le.ac.uk

***Abstract*—This paper proposes a solution to performance issues in the selection of Web services based on Quality of Service (QoS) that uses inference mechanisms from Semantic Web resources. Although several researchers highlight the benefits provided by the use of the Semantic Web techniques for searching and compose Web services with QoS, it can become costly when it depends on the approach adopted. Thus, we present a web service selection that uses QoS information in a replicated and distributed ontologies over different service providers. The results point to a significant improvement in the ontology inference process and thus makes the use of semantic resources viable in distributed systems to provide better QoS.**

***Keywords*-web services; semantic web; quality of service; performance;**

I. Introduction

Provide services with QoS is a differential in a globalized and competitive world. It is known that the number of Internet users is constantly growing due to the supply of devices such as *smartphones*, *tablets* and *netbooks* that allow users to access the Internet easily and more frequently.

Nowadays, companies make business among themselves (contractors and suppliers), enter into transactions with financial institutions (banks), and are even accountable to the government through applications running on the Web. However, this integration is only possible if there is a way to ensure the systems communication. The challenge to communicate between different applications written in different programming languages, and running on different platforms, raises interoperability problems, which can be mitigated through the use of Web services. However, there is still a challenge to choose which Web service will meet the QoS attributes desired by the client.

Semantic Web can be adopted to classify and select Web services based on their quality attributes. It provides information of a particular domain in an appropriate and expressive way. Generally, Semantic Web researchers use ontologies to represent the knowledge of a domain. Thus, when the ontology is well structured and formalized, it becomes a powerful knowledge base to store, manipulate and make inferences about the information.

Previous works [1], [2] and [3], demonstrated that the inference process requires a long time and expensive computational resources when large amounts of information is available. In this paper, we present improvements to selection process of Web services based on QoS, which are extremely important to both academy and industry researches fields. The main contributions of this paper can be summarised as: (1) provide a solution to the performance problems in ontologies inference process and (2) contribute with a new multidisciplinary approach for WS selection based on QoS attributes. A module called DP-WSOnto (Distributed and Parallel - Web Service Ontology) aims to fulfill these goals adequately. DP-WSOnto was initially introduced in a work in progress [4] and in this paper, other results and the development are better detailed.

The rest of this paper is structured as follows: Related work is discussed in the next Section II. In Section III the ontology and module used in this project are briefly described. Section IV presents the environment, planning and results analysis for the performance evaluation. Finally, Section V presents the conclusions and future work.

II. Related Work

Several studies propose Semantic Web services discovery seeking to increase the efficiency and response time. In [5], the author proposes the combination of the inference engine *Pellet DL reasoner* with the production rules engine named Jena. The purpose is to explore the reasoning capabilities of Pellet and also the production rules OWL 2 RL/RDF of Jena resulting in a faster implementation.

In [6] the authors proposed a Four-Level Matching Model for Semantic Web Service based on QoS Ontology. It was divided into: application domain matching, description matching of service, function matching of service and QoS matching. Despite, this paper has described detailed formal definition of all four types of matching. It has no performance evaluation of the proposed model. So, it is not possible to evaluate how fast and scalable this model is.

In [7] the authors used OWL as the ontology building language. Using the ontology modeling tools protege_3.4.4, they build a QoS ontology model which is extensible and hierarchical, divided into three levels: upper, middle and lower ontology. They proposed a Semantic Web Service Selection Algorithm based QoS Ontology (SWSSA), composed of four steps: firstly the service request vector takes semantic matching with service advertisement vector set. Secondly, they unify the measurement of QoS parameters. Thirdly, used the advertisement vector set to take value matching. Finally, they select the most adaptive service. In the paper, the authors executed some experiments, but they focused only on the users' satisfaction. No performance evaluation was made focusing on the average response time of the requests or the scalability of the proposed algorithm.

Another work [8] proposes a clustering-based algorithm using a vector space model. It uses WordNet resources to reduce the dimension of the term vector and to make semantic expansion to meet the user's request. Other approaches in this context are proposed in [9], and [10] that exploit clustering to filter and remove irrelevant research services, making this search process more efficient. Another way to improve the performance is proposed in [11] and it consists of parallelization of algorithmic reasoning. In that work, it was firstly proposed to partition the data set and also partition the rules set. It presents a parallel inference algorithm that uses the partition of rules and data to perform classification. This algorithm was more efficient and faster than sequential algorithms.

In this paper, we propose an inference process in parallel. We adopted the programming model ("Problem Decomposition") because this approach does not need major changes in algorithms and ontologies already implemented. Moreover, we use virtualization capabilities to ensure better performance and scalability.

III. Development

The development addresses the preparation of the ontology that is used by the DP-WSOnto module.

A. Ontology

The ontology proposed by [2] is used to validate the module. It includes elements of Web services with QoS context. Some of these elements are described as follows [2]:

- **Client:** *Clients* access and use the Web services, they have a QoS agreement with providers.
- **Provider:** The *Providers* include information from their Web services (functional and nonfunctional (QoS)) in the ontology, they also provide these Web services.
- **Services** The *Services* are Web services provided by providers.
- **Agreement:** *Agreements* are established between providers and clients.
- **QoS:** A *QoS* element stores all QoS attributes of a particular service or agreement.

Each element is represented by a class in the ontology and each class has subclasses, establishing a class hierarchy. The OWL provides relations among these classes (and subclasses) through properties that can also have characteristics.

In this way, the *Client* class is related to *Agreement* class indicating that the client has an agreement. The *Agreement* class is related to *QoS* class, it means a relation among an agreement to a specific QoS. Thus, these relationships can be semantically understood as: " *The client has an agreement and this agreement is related to a particular QoS*". Also, the *Provider* class is related to the *Service* class which is connected to the *QoS* class. These relations result semantically in the following: "*Providers have Web services that have a certain QoS*".

In addition, *Client*, *Agreement*, *Service* and *QoS* classes have subclasses that classify their elements as *Gold*, *Silver* and *Bronze* subclasses. The key issues are to determine the subclass to which a service belongs and how the values of the attributes in the *QoS* relate to the service. For example, if a service is related to a QoS which was inferred to be a *QoSGold* subclass, then the service will also be an element in the *ServiceGold* subclass. The same applies for the client element, because it will be a *ClientGold* only when its agreement is related to a QoS inferred as a subclass Gold (*QoSGold*). Therefore, at the time of entering information in the ontology, the providers do not need to determine which class the Web service is in, it is done by simply registering their services as a "generic" service related to one "Generic QoS ". Using data from this generic QoS, the inference engine can determine which subclass the new service will be part of. Likewise, it happens for the clients that only need to inform of the QoS values they wish in their agreement.

QoS attribute values of a particular service are nor always constant, neither has the value to exactly equal that required by the client. For this reason, a set of intervals for these values was adopted by using constraints. The idea of using data properties constraints to classify elements into "*Equivalent Classes*" comes from an example of another ontology in the literature, the Pizza Ontology, which is an example used in the tutorial from the University of Manchester. This tutorial shows an example of the use of equivalence class restrictions on data properties [12], in which pizzas are classified as *HighCaloriePizza* and *LowCaloriePizza* according to a property that indicates the amount of calories of each pizza.

A perceived problem in the work of [2] was that as more elements (individuals or instances) of the *Service* class were added to the ontology, more the inference time increased.

Thus, to investigate and compare this problem using the Pizza ontology, the authors also created more individuals (pizzas) in the ontology and noticed that the behavior was the same. This problem limits the number of Web services that could be registered in the ontology, because any changes in the ontology require a new inference process, which would increase the time in some cases.

B. DP-WSOnto Module

The DP-WSOnto (Distributed and Parallel - Web Service Ontology) [4] module searches Web services with QoS in parallel. When module starts, the ontology inference process is executed by an inference machine (reasoner) called Pellet. The Pellet aims to analyze, verify and make inferences on the ontology elements. It is a complete inference engine and widely used. However, in those cases that the ontology uses "*Equivalent Classes*" restrictions, the inference processing is performed in a single processing core. During the investigation of the complex source code, it was noticed the existence of several recursive calls, making it difficult to be parallelized. Thus, the code for the inference process was not changed. However, a parallel approach was adopted through threads creation. The number of threads should be equal to the number of providers, because each thread reads and infers the ontology of a specific provider. So, the previous ontology (within various services of several providers) was copied to other ontologies, each one containing only the services of a single provider. Furthermore, a single thread was used per core due performance reasons. Figure 1 shows the flow adopted for this approach.

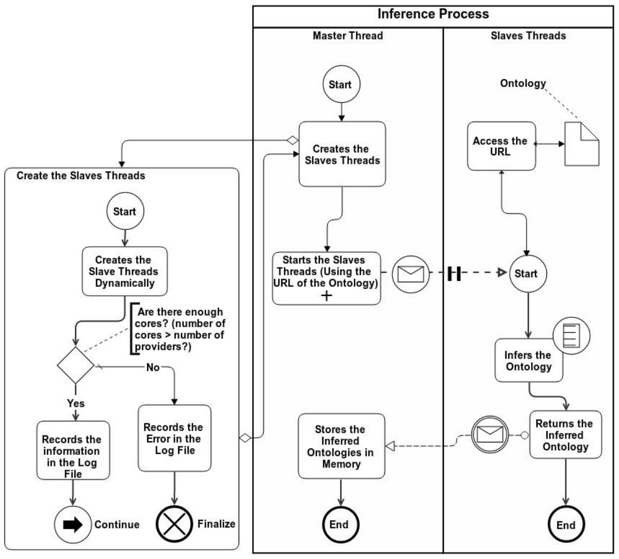
Inference Process
Master Thread
Slaves Threads
Create the Slaves Threads
Start
Creates the Slave Threads Dynamically
Are there enough cores? (number of cores > number of providers?)
Yes
No
Records the information in the Log File
Records the Error in the Log File
Continue
Finalize
Start
Creates the Slaves Threads
Starts the Slaves Threads (Using the URL of the Ontology)
Stores the Inferred Ontologies in Memory
End
Ontology
Access the URL
Start
Infers the Ontology
Returns the Inferred Ontology
End

Figure 1. Inference Process Flow.

In this scenario, considering the existence of several inferred ontologies into DP-WSOnto, it is presented a new parallel approach for the process of searching for Web services with QoS. In this new approach, multiple instances of the search algorithms are started in parallel, each thread receives three arguments: the ontology URL, an OntModel object used to store the ontology inferred and a CyclicBarrier object used by all threads to synchronize the process. Thus, each algorithm instance reads one of the previously inferred ontologies and at the end of the execution of all threads, a unique and composed result is stored in memory to be used by the client (Figure 2). The proposal exploits the use of parallelism through decomposition of data (number of services), and several threads are created to read data from a distributed ontology.

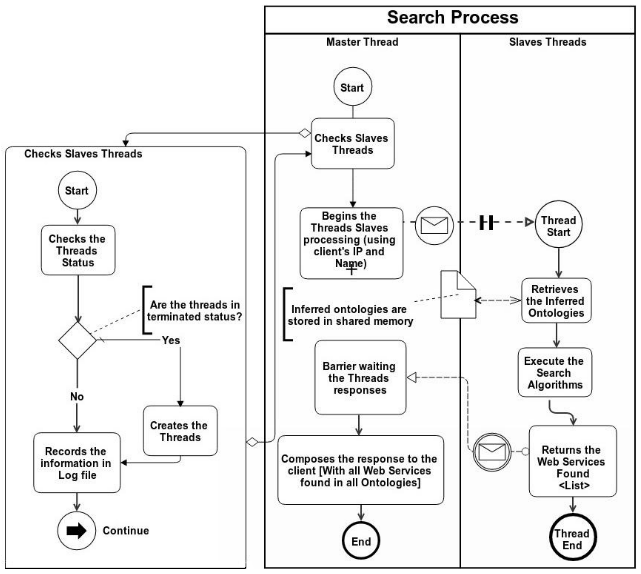
Search Process
Master Thread
Slaves Threads
Checks Slaves Threads
Start
Checks the Threads Status
Are the threads in terminated status?
Yes
No
Creates the Threads
Records the information in Log file
Continue
Start
Checks Slaves Threads
Begins the Threads Slaves processing (using client's IP and Name)
Inferred ontologies are stored in shared memory
Barrier waiting the Threads responses
Composes the response to the client [With all Web Services found in all Ontologies]
End
Thread Start
Retrieves the Inferred Ontologies
Execute the Search Algorithms
Returns the Web Services Found <List>
Thread End

Figure 2. Search Process Flow.

IV. Performance Evaluation

A. Environment Configuration

The computing infrastructure used in the performance evaluation is described in Table I. The use of virtual machines is justified because the virtualization makes the machines reconfiguration flexible and faster, this is a technique widely used in cloud computing today.

Table I
Hardware Elements

Elements	Components	Features
Real Machine for Server Module	CPU (12 cores) RAM Memory and Disk	Intel(R) X5660 @ 2.80GHz 12 GB and 500 GB
Virtual Machine for Server Module	CPU (8 cores) RAM Memory and Disk	Intel(R) X5660 @ 2.80GHz 8 GB and 21 GB
Real Machine for Client and Database	CPU (4 cores) RAM Memory and Disk	Intel(R) i3-2100 @ 3.10GHz 4 GB and 1TB
Virtual Database Server (MySql)	CPU (1 core) RAM Memory and Disk	Intel(R) i3-2100 @ 3.10GHz 1 GB and 8 GB
Virtual Client Desktop	CPU (1 core) RAM Memory and Disk	Intel(R) i3-2100 @ 3.10GHz 1 GB and 8 GB

Besides the hardware elements of Table I, five real machines running the Apache Web server and providing replicated ontologies were used as providers. The list of software elements and their uses in the experiments are described in Table II.

Table II
SOFTWARE ELEMENTS

Elements	Utilization	Version
Linux Debian	Real Machine (Module)	2.6.32-5-amd64
Linux Ubuntu	Real Machine (Client and DB)	3.0.0-26-generic
Linux Ubuntu	Virtual Machines	3.2.0-29-generic
Hypervisor	KVM - QEMU emulator	qemu-kvm-0.14.1
Apache Web Servers	Provide ontologies via URL	2.2.14
Apache Tomcat	Host the Web services	6.0.26
Apache Axis2	SOAP messages	1.4.1
jUDDI	Web services Repository	0.9rc4
JVM	All components	1.6.0_22
MySQL Server	Store performance results	5.1.41-3
Pellet	Reasoner	2.2.2
Jena	Create algorithms with Semantic	2.6.3
Log4J	Record logs (Error, Trace, etc.)	1.2.16

B. Experiments Design

The experiment design should provide information on two main points: (1) evaluate the performance gain when the ***inference process*** is conducted in parallel using copies of the ontology in a distributed scenario and, (2) evaluate the performance gain in the ***search process*** for services with QoS when it executes algorithms in parallel over the DP-WSOnto. Therefore, in the scope of this work two experiments designs are considered. The first one conducts a performance comparison between sequential and parallel inference process approaches. Table III shows the factors and levels related to this Experimental Design (ED) which was called as **ED-Ontology**.

Table III
FACTORS AND LEVELS RELATIVE TO ED-ONTOLOGY

Factor	Levels	Description
Number of Services	**300, 600, 900 and 1200**	Number of Services (individuals) registered in ontologies, divided equally among the five providers and replicated ontologies.
Approach	**Sequential and Parallel**	Approach adopted in the inference process.

The second design is related to the search process of Web services with QoS which is executed by the DP-WSOnto algorithms. The search algorithm will perform the search process that was described in Section III-B and is executed only after the module initialization. In other words, the search algorithm will perform the search as the ontologies have been inferred. Therefore, these are distinct activities that occur at different times. Table IV shows the factors and levels related to this experimental design that was called as **ED-Search**.

For the **ED-Ontology**, ten repetitions were performed for the eight possible combinations. For the **ED-Search**, twenty

Table IV
FACTORS AND LEVELS RELATIVE TO ED-SEARCH

Factor	Levels	Description
Number of Services	**300, 600 and 1200**	Number of Services in the ontologies.
Approach	***Sequential* e *Parallel***	Search approaches.

repetitions were made for the twelve possible combinations. A 95% confidence level was adopted for both experiments designs. The response variable adopted for **ED-Ontology** is the average time spent on the inference process. For **ED-Search**, the response variable is the average response time (RT). The RT is the time spent from sending the request until the arrival of the response in the client. Thus, RT includes the network traffic time, search process in the ontology and also the marshalling and unmarshalling of SOAP messages.

C. Result Analysis

1) ***Inference Process - (DE-Ontology):*** Detailed results of the **ED-Ontology** are available in Table V, which shows the Experiment (Exp.), Average Time (AT) of the inference process, Standard Deviation (SD), Confidence Interval (CI) and Maximum (MAX) and Minimum (MIN) intervals (calculated from average time and confidence interval).

Table V
ED-ONTOLOGY RESULTS (SECONDS)

Exp.	AT	SD	CI	Int. MAX : MIN
Seq-300	35.98227	0.69766	0.43240	36.41467 : 35.54987
Para-300	7.94248	0.22382	0.13873	8.08121 : 7.80375
Seq-600	136.39977	3.29345	2.04126	138.44103 : 134.35851
Para-600	13.57439	0.34926	0.21647	13.79086 : 13.35792
Seq-900	313.13576	4.52627	2.80536	315.94112 : 310.33040
Para-900	22.45356	0.48272	0.29919	22.75275 : 22.15437
Seq-1200	543.42814	9.21674	5.71249	549.14063 : 537.71565
Para-1200	35.68345	0.59537	0.36901	36.05246 : 35.31444

The approach *Parallel* has a shorter time to infer ontologies. An improvement in that time was an expected result, but the enhancement in processing speed was surprising. For example, the parallel approach for 300 services was about 4.5 times faster than the sequential approach; with 1200 services, the difference was approximately 15 times. These results make the inference process feasible for those cases where there is a need for a faster response or where the ontology needs to be constantly updated. Thus, the negative effect of *caching* can be minimized (caching is working with an outdated ontology).

The ***speedup*** was calculated to validate and measure the benefit gained by parallelization. Therefore, the values of speedup obtained for 300, 600, 900 and 1200 services are (4.5314), (10.0508), (13.9478), and (15.2303) respectively. The efficiency [13] for 300, 600, 900 and 1200 services are respectively: (0.9062), (2.0101), (2.7895), and (3.04606). For 300 services, this ratio was less than 1 because the overhead of our approach (division of tasks, process and retrieval of results) takes more time than the sequential processing of the ontology.

*2) **Search Process - (DE-Search)**:* These results follow the same tendency, the more services registered in the ontologies (higher load) the longer the response time (on average). This indicates that the *overhead* generated by the threads creation and execution of parallel approach has not overcome the sequential execution with 300 services. In the other possible cases, the parallel approach was always better than the sequential approach. Table VI shows the ***speedup*** and ***efficiency*** results (number of threads: $p = 5$).

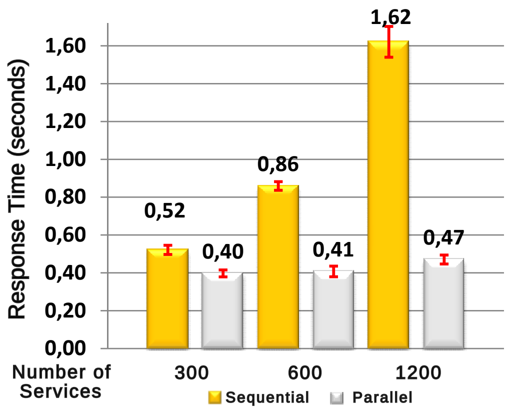

Figure 3. Response Time (RT) Results.

Table VI
Speedup AND *Efficiency* ($p = 5$) (ED-SEARCH)

Experiment	Speedup (RT)	Efficiency (RT)
300-SPARQL	1.3128	0.2625
600-SPARQL	2.1080	0.4216
1200-SPARQL	3.4573	0.6914

V. Conclusion and Future Work

This paper presented a solution to improve the performance in the inference process of ontologies for Web Services selection with QoS, especially those that consider the use of constraints "*Equivalent Classes*". The inference process time had a great improvement and the use of virtualization resources also contributed to obtain a better performance and scalability. Moreover, the benefit provided by this research also can be adopted by different branches of researches that use ontologies and require inference processes.

In future works, there are plans to perform new performance evaluations considering other ontologies and using other inference engines comparing our results with other studies of the literature.

Acknowledgments

The authors thank the Brazilian Foundation FAPESP and PIPECT/IFSP for the financial support given to this work.

References

[1] J. Zhang, X. Yu, P. Liu, and Z. Wang, "Research on improving performance of semantic search in uddi," in *Intelligent Systems, 2009. GCIS '09. WRI Global Congress on*, vol. 4, may 2009, pp. 572 –576.

[2] L. Nakamura, J. Estrella, M. Santana, and R. Santana, "Semantic web and ontology applied to web services discovery with qos," in *Sistemas Computacionais (WSCAD-SSC), 2011 Simpósio em*, oct. 2011, p. 5.

[3] C. Surianarayanan and G. Ganapathy, "A survey on optimization approaches to semantic service discovery towards an integrated solution," *ICTACT Journal on Soft Computing*, vol. 2, no. 4, pp. 377–383, 2012.

[4] L. Nakamura, P. do Prado, R. De O Libardi, L. Nunes, J. Estrella, R. Santana, M. Santana, and S. Reiff-Marganiec, "Fast selection of web services with qos using a distributed parallel semantic approach," in *Web Services (ICWS), 2014 IEEE International Conference on*, June 2014, pp. 680–681.

[5] G. Meditskos and N. Bassiliades, "Dlejena: A practical forward-chaining owl 2 rl reasoner combining jena and pellet," *Web Semant.*, vol. 8, no. 1, pp. 89–94, Mar. 2010.

[6] G. Guo, F. Yu, Z. Chen, and D. Xie, "A four-level matching model for semantic web service selection based on qos ontology," in *Information Science and Engineering (ISISE), 2010 International Symposium on*, Dec 2010, pp. 630–634.

[7] W. Junhao, G. Jianan, J. Zhuo, and Z. Yijiao, "Semantic web service selection algorithm based on qos ontology," in *Service Sciences (IJCSS), 2011 International Joint Conference on*, May 2011, pp. 163–167.

[8] H. Gao, S. Wang, L. Sun, and F. Nian, "Hierarchical clustering based web service discovery," in *Service Science and Knowledge Innovation*, ser. IFIP Advances in Information and Communication Technology, K. Liu, S. Gulliver, W. Li, and C. Yu, Eds. Springer Berlin Heidelberg, 2014, vol. 426, pp. 281–291.

[9] S. Han, H. Wang, and L. Cui, "A user experience-oriented service discovery method with clustering technology," in *Computational Intelligence and Design. ISCID '09. Second International Symposium on*, vol. 2, dec. 2009, pp. 64 –67.

[10] J. Liu, K. He, J. Wang, F. Liu, and X. Li, "Service organization and recommendation using multi-granularity approach," *Knowledge-Based Systems*, vol. 73, no. 0, pp. 181 – 198, 2015.

[11] R. Soma and V. Prasanna, "Parallel inferencing for owl knowledge bases," in *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, sept. 2008, pp. 75 –82.

[12] K. Holger, A. Rector, R. Stevens, C. Wroe, S. Jupp, G. Moulton, and N. Drummond, "A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.2," 2009.

[13] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing (2nd Edition)*. Pearson Addison Wesley, 2003.