CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (in press)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1204

Performance analysis of a semantics-enabled service registry

W. Fang, S. Miles and L. Moreau*,†

School of Electronics and Computer Science,
University of Southampton, Southampton, U.K.

SUMMARY

Service discovery is a critical task in service-oriented architectures. In this paper, we study GRIMOIRES, the semantics-enabled service registry of the OMII software distribution, from a performance perspective. We study the scalability of GRIMOIRES against the amount of information that has been published into it. The methodology we use and the data we present are helpful for researchers to understand the performance characteristics of the registry and, more generally, of semantics-enabled service discovery. Based on this experimentation, we claim that GRIMOIRES is an efficient semantics-aware service discovery engine. Copyright © 2007 John Wiley & Sons, Ltd.

Received 31 May 2006; Revised 4 December 2006; Accepted 10 February 2006

KEY WORDS: Web service; service discovery; service registry; metadata annotation

1. INTRODUCTION

In service-oriented architectures, service discovery is a critical task underpinning service invocation, service orchestration, and service monitoring. A service registry is intended to be the contact point where service descriptions can be published and services suitable for invocation can be discovered based on the previously published information.

Among the many proposals for service registries in this context [1–3], the Universal Description, Discovery, and Integration (UDDI) specification [1] is the standard for Web service publication and discovery. UDDI defines both a data model to describe services and a set of interfaces to publish and discover service descriptions. However, UDDI suffers from some limitations that hinder its

*Correspondence to: Luc Moreau, School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, U.K.
†E-mail: l.moreau@ecs.soton.ac.uk

Contract/grant sponsor: EPSRC; contract/grant number: GR/S90843/01

widespread adoption. UDDI offers no mechanism to refer to a service interface signature (such as operations or input and output messages) and therefore is unable to discover a service according to such a signature.

Furthermore, registering only service interfaces does not necessarily lead to satisfactory service discovery. Indeed, a service interface such as a WSDL document usually characterizes the messages accepted by a service, the type of data they must contain, and the way they must be encoded. On the other hand, users are interested in the behavior or function of a service, usually expressed according to high-level concepts, over which reasoning can take place. For the purpose of this paper, the former kind of information is referred to as *syntactic-level* description, whereas the latter is denoted as *semantic-level* information. Hence, it is desirable to enrich existing syntactic-level description with semantic-level information, which allows users to express more powerful and accurate queries. However, UDDI lacks the capability of annotating service descriptions with structured metadata; in this context, we define metadata as any information that is not captured in the UDDI service description model. Such metadata can be arbitrary, but in particular, can include semantic-rich information to enhance service discovery.

Having identified that metadata annotation can play a vital role in assisting service discovery, we designed and implemented GRIMOIRES, a UDDI-compliant service registry extended with powerful metadata attachment support (http://www.grimoires.org). Adopting the UDDI standard facilitates the acceptability of the GRIMOIRES approach by the Web service community. In addition, GRIMOIRES supports metadata attachment to service descriptions by service providers and consumers, as well as third parties [4]. Further descriptions of services, such as a particular database that a service will use or the semantic type of the input of a service operation, can be added as annotations to facilitate future service discovery. Furthermore, GRIMOIRES relies on the Resource Description Framework (RDF) to encode all information it contains. RDF [5], a language for representing information about resources in the World Wide Web, constitutes the foundational layer of the Semantic Web [6]. Since it supports ontology-based reasoning, information registered in GRIMOIRES can be reasoned over, potentially using ontology models, to support *semantic discovery* where information is discovered according to inference, rather than syntactic matching. By exploiting GRIMOIRES' metadata attachment capability, domain-specific service annotation and service discovery by interface signature can be accomplished, as demonstrated in the bioinformatics e-Science project myGrid (http://www.mygrid.org.uk) [7]. GRIMOIRES is now adopted by the Open Middleware Infrastructure Institute (http://www.omii.ac.uk) software distribution as its default service registry.

In this paper, we study the GRIMOIRES approach from the perspective of performance. In particular, we analyze the scalability of GRIMOIRES against the amount of information published into it. We have conducted extensive experiments to study the performance of GRIMOIRES: (1) GRIMOIRES' performance using various persistent stores; (2) the memory usage of GRIMOIRES; (3) GRIMOIRES' performance under the security framework specified by WS-Security [8] and related specifications; (4) the performance comparison between GRIMOIRES and jUDDI [9], an open source standard UDDI registry, to study the overhead of representing published information in the RDF format; (5) the performance of metadata annotation and metadata-based discovery; and finally (6) GRIMOIRES' throughput under simultaneous requests.

The rest of the paper is organized as follows: in Section 2 we discuss related work; in Section 3 we present GRIMOIRES' approach towards semantics-enable service publication and discovery; in Section 4 we introduce our methodology to investigate the registry's performance; in Section 5 we present the experiment results and its detailed analysis; and in Section 6 we conclude the paper and discuss future work.

2. RELATED WORK

The UDDI service registry [1] is the standard for Web service discovery. Service descriptions in UDDI are composed from a limited set of high-level data constructs: Business Entity is the data model for service providers, Business Service for services themselves, Binding Template for the concrete bindings of the services, and Technical Model (tModel) for some shared knowledge such as a category system or the technical interfaces of services. However, UDDI provides only limited support to annotate a published service, or to discover a service based on annotated metadata. While UDDI provides ways to attach a pair of key and value as a form of metadata to a service description, UDDI lacks the capability of annotation of and inquiry by structured metadata. UDDI also does not allow service consumers or third parties to attach metadata to a published service description. This implies that a service description is the publisher's view of a service only. The WSDL language is the standard way to define the technical interface of a Web service. In UDDI, tModels are used to link a WSDL interface to a service. However, UDDI does not provide users with the capability to annotate an element of its technical interface, or to search for a service based on a certain characteristic of its technical interface. For instance, UDDI does not allow users to qualify the input of an operation of a Web service with a semantic type. Such a semantic type would provide a description of the input in terms of the service's semantics rather than the type encoded in the SOAP message [7]. In the absence of such a facility, users have to express queries that do not refer to the service's semantics but to its syntactic level.

There exists other service description models than UDDI. For instance, OWL-S [3], formerly DAML-S, uses the OWL Web ontology language to describe a Web service from three component perspectives: what it does (service profile), how it works (service model), and how to access it (service grounding). BioMOBY [10] defines a service as an atomic process or operation that takes a set of inputs and produces a set of outputs. In BioMOBY, the service, inputs, and outputs can all take semantic types; inputs and outputs also have syntactic types. The ebXML registry [2] is designed to be a generic information management system. It defines both a registry information model and a set of APIs. The ebXML registry provides a Web service profile that specifies the conventions to make the ebXML registry act as a Web service registry. In GRIMOIRES, we take the Web service registry approach, and extend it whenever necessary. By exploiting standards such as UDDI, we can disseminate our ideas to a potentially large user community.

The Globus Toolkit (http://www.globus.org) provides the Monitoring and Discovery System (MDS), consisting of a suite of Web services, to monitor and discover resources and services on Grids. MDS focuses on the mechanism to disseminate and gather information on Grids rather than the information model to describe services or resources. Information is published in XML according to some schema. While the GLUE schema [11] is used for compute information, the owners of information sources or Grid resources have the freedom to define their own schema, so that arbitrary XML data can be published to describe service profiles and states. As far as inquiry is concerned, MDS relies on the XPath language. One possible integration between MDS and GRIMOIRES could be the use of MDS as an automatic service information collector for the GRIMOIRES registry.

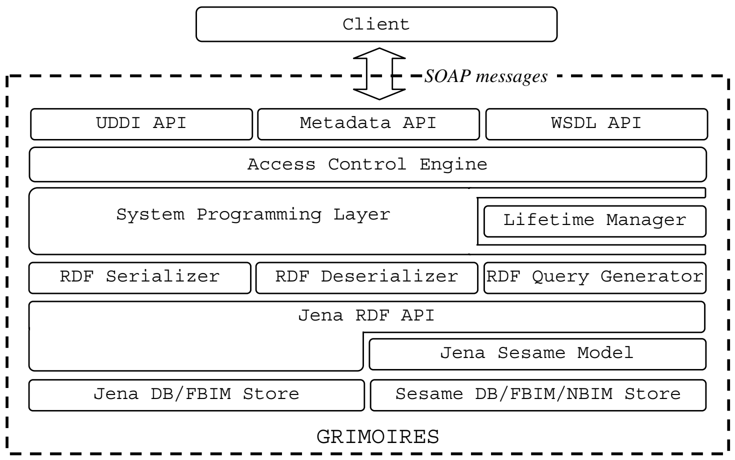

Figure 1. The architecture of GRIMOIRES.

3. GRIMOIRES APPROACH

3.1. Architecture

GRIMOIRES is a UDDI compliant registry for Web services, which itself is implemented as a Web service, shown in Figure 1. Clients interact with GRIMOIRES through sending and receiving SOAP messages. In addition to the UDDI interface, GRIMOIRES also provides other interfaces: the metadata interface allows clients to publish and inquire over metadata, and the WSDL interface caters for WSDL descriptions which can be published and inquired over by clients.

An access control layer within GRIMOIRES enforces fine-grained access control for each published entity. All requests coming through the Web service interfaces reach the policy decision point of the access control engine before they are passed to the system programming layer, if allowed.

The system programming layer (SPL) implements the business logic of the UDDI/Metadata/WSDL interfaces. The SPL calls the RDF Deserializer/Serializer to deserialize/serialize UDDI/Metadata/ WSDL descriptions to and from RDF triples, which reside in a triple store. RDF statements are formed of triples (subject–property–object) and are persistently stored in a triple store. The SPL also calls the RDF Query Generator to translate UDDI/Metadata/WSDL queries to RDF queries. The lifetime manager is in charge of removing the entities when their respective lifetimes expire.

GRIMOIRES can use either Jena [12] or Sesame [13] RDF triple store implementations. The triple store can take the form of a relational database (DB), or an in-memory store with persistence support. It gives deployers options to balance scalability with performance.

In order to simultaneously achieve persistence and good performance we introduce a double store configuration of GRIMOIRES, which is called *native backup in-memory* (NBIM). In NBIM, two triple stores are leveraged: an in-memory store that keeps all triples in memory for efficient access and a native store that implements B-Trees on the file system. Upon publication, information is added to both the in-memory store and the native store, whereas queries are run over the in-memory store; hence, the in-memory store is used for performant access, whereas the native store offers persistence.

3.2. Functionality

Other than the compliance with the UDDI specification, GRIMOIRES has the following features from the functionality perspective.

3.2.1. Publication and discovery of service interface descriptions

GRIMOIRES has the ability to publish and inquire over WSDL documents describing the technical interfaces of services. By registering WSDL documents, GRIMOIRES is able to support service discovery by interface signature.

3.2.2. Metadata annotation

Metadata annotation provides further information about service descriptions that are difficult to express in the predefined service description data model. Currently, entities to which metadata can be attached are the UDDI Business Entity, Business Service, Technical Model, Binding Template, WSDL operation and message part, as well as metadata.

A piece of metadata is in the form of an RDF triple: the subject is the entity to be annotated, the predicate is the type of the annotation relationship, and the object is the annotated value. The metadata value can be a string, a URI referring to some predefined ontology concept, or structured data in RDF for more complex property description. For example, a RDF triple (aService, useDatabase, aPostgreSQL) attaches a metadata to a service, to describe that a service is using a certain PostgreSQL database.

There is no limit to the number of metadata attachments each entity can have. Each piece of metadata can be updated and deleted independently with regard to the owner entity and other metadata attached to the same entity.

3.2.3. Inquiry by metadata

The goal to support metadada annotation is to allow service discovery by metadata, which enables services to be discovered in a user-specified way. In GRIMOIRES, an entity can be found according to its attached metadata which is expressed as either a sequence of metadata type and value pairs or an RDF query statement. To support queries over both metadata and information in the UDDI data model (such as the name of a service), we have extended the UDDI service finding operation with similar metadata query facility.

Inquiry by metadata is done by string matching when ontology support is not required. When ontology support is needed, inquiry by metadata leverages ontology reasoning capability provided by the underlying RDF triple store.

3.2.4. Third-party annotations

The ability to publish metadata is available to service providers, service consumers, and also third parties. This provides the flexibility of allowing users with expert knowledge to enrich service descriptions in ways that might not be conceivable to the original publishers. For instance, users can provide their personal ratings on services as metadata attachment, which can then be taken into consideration by other service users. Supporting third-party annotations requires a proper authentication and authorization framework in place, which is discussed in the following section.

3.2.5. Signature-based authentication

UDDI v2 and v3 specifications rely primarily on the use of authentication tokens to authenticate users for publisher API calls. In implementations such as jUDDI [9], this is generally achieved through a username/password credential scheme. However, this authentication method does not scale well for Grid environments, which typically use certificate-based authentication schemes. The OMII framework provides an implementation of SOAP message signing and verification in accordance with WS-Security standards [8]. When deployed within the OMII container, GRIMOIRES can extract the Distinguished Name (DN) from the submitted X509 client certificate for authentication purposes. Incorporating signature usage in this way makes it easier to integrate GRIMOIRES into existing Grid security infrastructures, as well as providing an important building block for single sign-on capabilities, an important requirement for many Grid applications.

3.2.6. Access control

Access control is on the basis of authenticated identity, and is applied at the granularity of each registered entity, for example a service, a WSDL file, or a piece of metadata. The access control engine needs to decide which user can perform which operation on which entity. The access control assertions are represented as metadata and are attached to the corresponding data entries.

3.2.7. Lifetime management

In GRIMOIRES, each entity that has a known limited lifetime can be attached with a special metadata denoting its scheduled termination time. Proper actions will be taken by the registry to entities whose scheduled termination time arrives. Equipped with lifetime management, GRIMOIRES is made more suitable for the Grid environment, where transient services may be used extensively.

To efficiently identify out-of-date entities, a binary tree sorted in an ascending order of termination time is maintained to hold all entities scheduled with a termination time. A thread periodically compares the entities with the latest termination time against the current time. Having been identified, the out-of-date entities can be simply deleted or hidden from discovery according to user provided policies.

4. EXPERIMENT METHODOLOGY

In a service registry, two fundamental operations are service publication and service discovery. It is interesting to know the overhead of individual publication and discovery operations with respect to the data size of the registry. We want to investigate to what extent the publication and discovery overheads are affected when an increasing amount of data are registered into GRIMOIRES.

In this test, we use the UDDI data model to describe a Web service. The corresponding UDDI APIs are invoked to publish a service and to discover a service. Each UDDI API invocation is a Web service invocation which involves a pair of SOAP messages (request and response).

In the test, we repeat the following procedure: (1) publish 100 different service descriptions, and (2) among all the service descriptions currently registered in GRIMOIRES, randomly inquire for 100 service descriptions. We then calculate the average overhead of publishing one service description and inquiring for one service description under the corresponding registry data size.

Using the above method, we measure the service publication and discovery performance of GRIMOIRES against the registry data size under various settings, to investigate the performance scalability of GRIMOIRES with respect to the number of services published. We also apply this methodology to study the performance of metadata attachment and query by metadata.

5. PERFORMANCE ANALYSIS

Except where stated otherwise, the platform on which these tests are conducted is a PC with an Intel P4 3 GHz CPU. GRIMOIRES runs as a Web service deployed in a Web service container such as an OMII container or an Apache Axis. In both cases, JDK 1.5 is used with the heap size set to 1 GB. In these experiments, we follow the methodology described in the previous section.

5.1. Performance with various persistent stores

It is important for the registry to be persistent, so that the contents of the registry is not lost at the moment of a crash. The persistence of GRIMOIRES relies on its RDF triple store. Currently, GRIMOIRES supports the following persistent stores: a Jena store using PostgreSQL as backend, a Jena store using Berkeley DB Java Edition (BDB JE) as backend, a Sesame file-backed in-memory (FBIM) store, and a NBIM double store configuration. In this experiment, we study GRIMOIRES' performance with various persistence stores, as shown in Figures 2 and 3 for the publication and inquiry performance, respectively. Table I shows the time to publish and inquire for a service when there are 2000 services in the registry for various triple store configurations. Note no matter which store is used in GRIMOIRES, GRIMOIRES presents clients with the equivalent publication and inquiry capability.

PostgreSQL is an open source object-relational database management system widely used in production environments. Rather than a relational database engine, BDB JE is a data store that can be embedded in applications. PostgreSQL is more efficient than BDB JE for inquiry, but slower than BDB JE for publication. An alternative scheme for GRIMOIRES to provide data persistence is to use a FBIM store, where triples reside in memory for better retrieving performance, and are synchronized to a file for persistence after a variable and adjustable delay. In the test, the delay is set to 200 ms, which means in general the published data will be made persistent on a file after 200 ms since its publication. However, if there is a new publication before the pending data is written to file, the timer for the synchronization delay will be reset. NBIM is the best in terms of performance among all configurations. In NBIM, the published information is immediately made persistent to the native store.

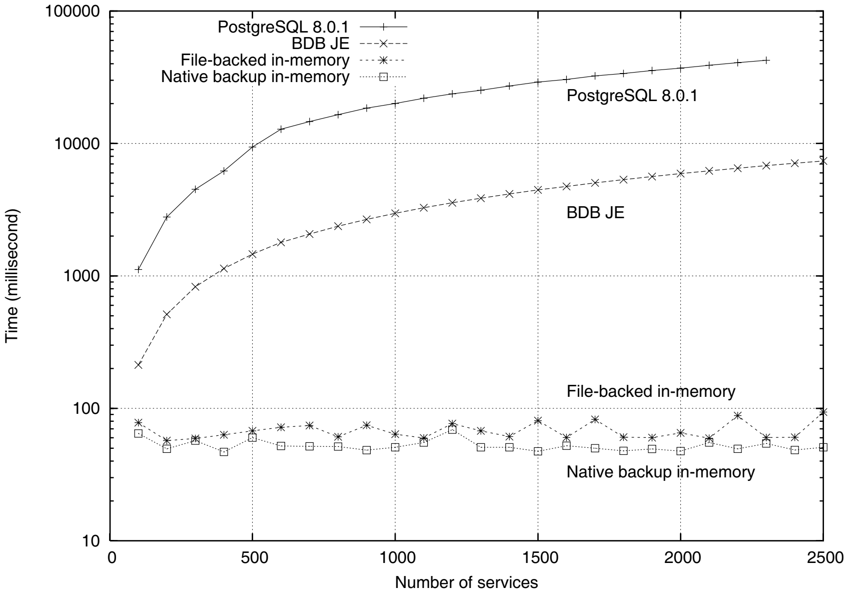

Figure 2. Publication overhead (persistent stores).

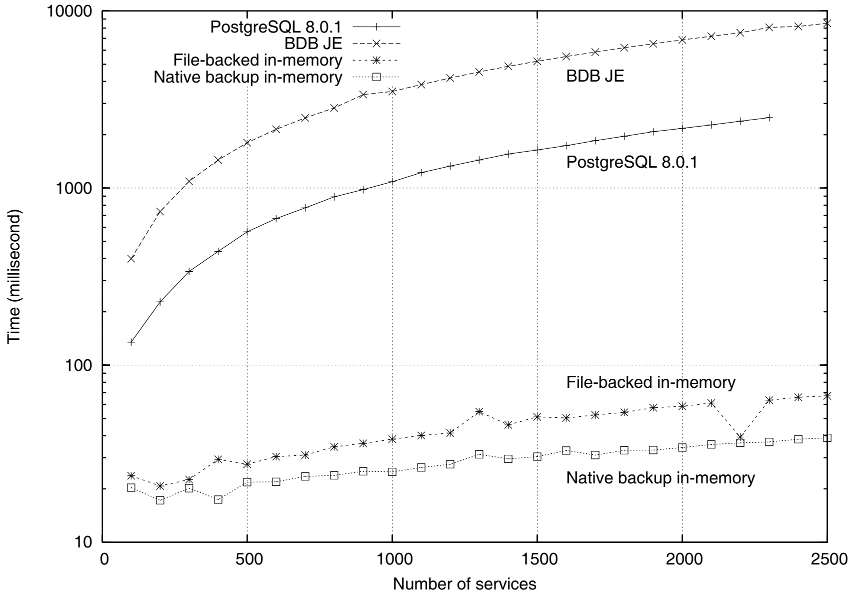

Figure 3. Inquiry overhead (persistent stores).

Table I. The time (in milliseconds) to publish and inquire for a service when there are 2000 services in the registry.

	Publish	Inquire
PostgreSQL	37 066.3	2172.8
BDB JE	5919.9	6862.4
File-backed in-memory	65.28	58.57
Native backup in-memory	47.55	34.24

NBIM combines the good persistence of the native store at the time of publication and the good performance of the in-memory store at the time of inquiry.

However, the scalability of FBIM and NBIM is limited by the memory size. In the circumstances where the amount of data to be published might exceed the memory size, other persistent stores have to be used in GRIMOIRES.

5.2. Memory usage

In this experiment, we study the memory usage behavior of GRIMOIRES with an in-memory store. By doing this, we investigate the effect in terms of memory consumption, of registering a large amount of data into GRIMOIRES with an in-memory store. We study the FBIM store with a synchronization delay of 200 ms, but the result can also be applied to the NBIM configuration.

Figure 4 shows the heap usage of GRIMOIRES' business logic when using the FBIM RDF triple store. It requires about 3.4 MB for 100 services, and 474 MB memory for 20 000 services. Publishing one service adds 138 RDF statements to the triple store, which consumes up about 25 kB of memory. We argue that this memory requirement can easily be satisfied on today's servers or even PCs, and the number of services contained in this amount of memory can meet the requirement of most environments. For example, the number of services available to use through the myGrid portal is in the scale of thousands.

Figure 5 shows the average number of minor garbage collections incurred per service publication when using the FBIM triple store. Sun JDK 1.5 uses a generational garbage collector by default. A minor collection collects the young generation only, whereas a major collection also collects the tenured generation. When there is enough allocatable memory in the heap, a typical minor collection is of the order of a few milliseconds, and a typical major collection is of the order of tens of milliseconds to one second. During the experiment, there are only 26 major collections, 18 of which occurred during the very beginning part of the experiment. As seen in Figure 5, when there are more than 3000 services in the registry, there will be one minor garbage collection per two service publications on average. Comparing the overhead of a minor collection with that of service publication and inquiry, it can be observed that the disruption caused by garbage collection is not serious under the condition that there is enough allocatable memory in the heap.

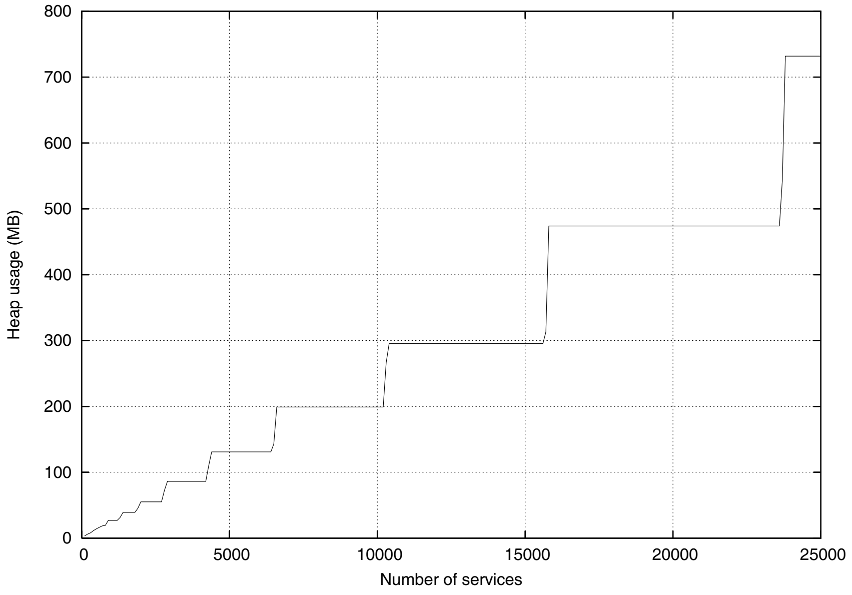

Figure 4. Heap usage.

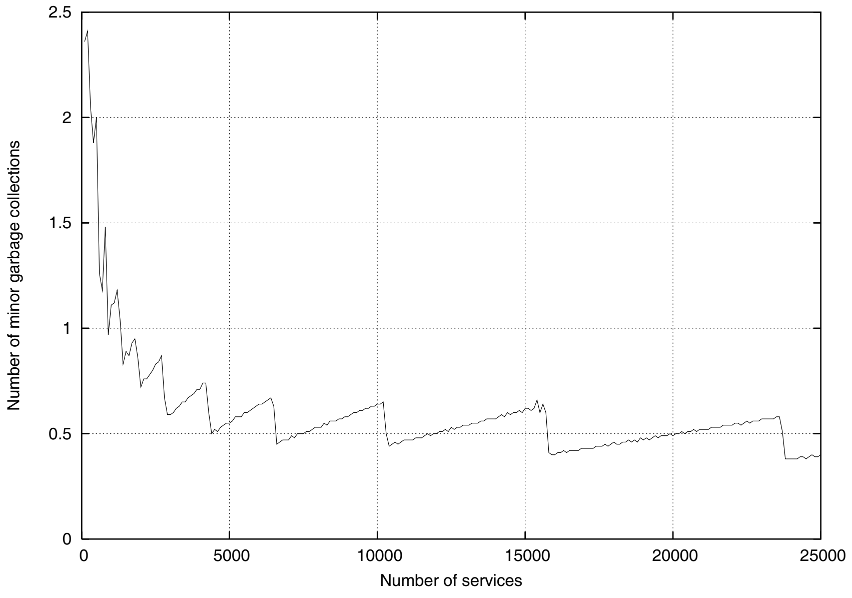

Figure 5. Average number of minor garbage collections.

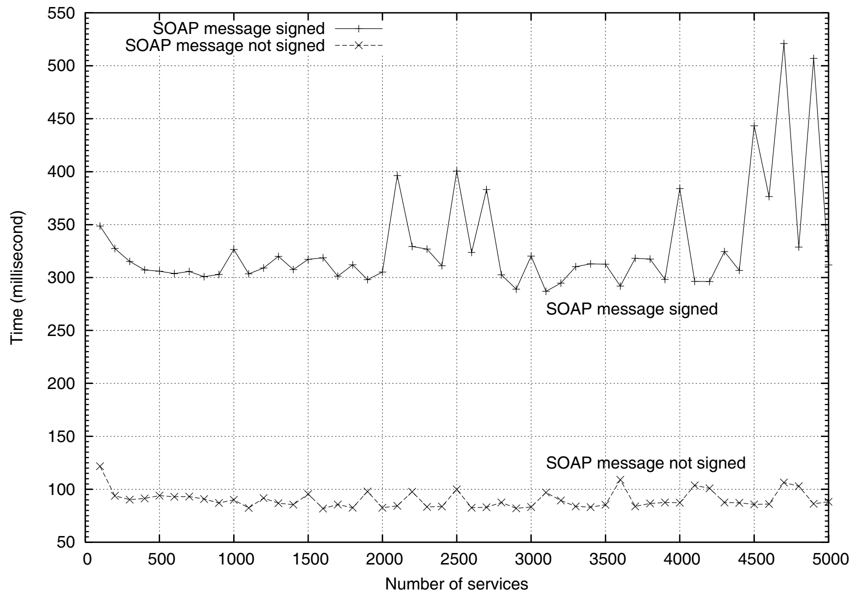

Figure 6. Publication overhead (WS-Security).

5.3. Performance with WS-Security

In a further experiment, we measure GRIMOIRES' performance under the framework of WS-Security [8]. The OMII environment provides the functionalities of creating X.509v3 certificate based signatures on outgoing client-side SOAP messages, and verifying these signatures on server-side Web services deployed in the OMII container. Signatures are created and verified through the client-side and server-side handlers, respectively, which intercept SOAP messages and process them in a way that is transparent to the client and the service.

We deploy GRIMOIRES in OMII 2.3.3 container that runs on a computer using an Intel P4 3 GHz CPU and 1 GB memory. In this test, GRIMOIRES is configured to use the FBIM store with a synchronization delay of 200 ms. We measure its performance with signing of SOAP messages both disabled and enabled. Figure 6 shows the publication overhead, and Figure 7 shows the inquiry overhead. As seen from the figures, the signing and verifying of SOAP messages introduce a significant overhead.

Table II shows the time to publish and inquire a service when there are 2000 services in the registry. As seen in the table, the signing of SOAP message has a more significant impact on the publication overhead than on the inquiry overhead. This is because there are three Web service interactions involved during the publication, and only two during the inquiry. The overhead due to SOAP message signing during the publication, i.e. the difference between when signing is enabled and when signing is disabled, is roughly 1.5 times of that during the inquiry.

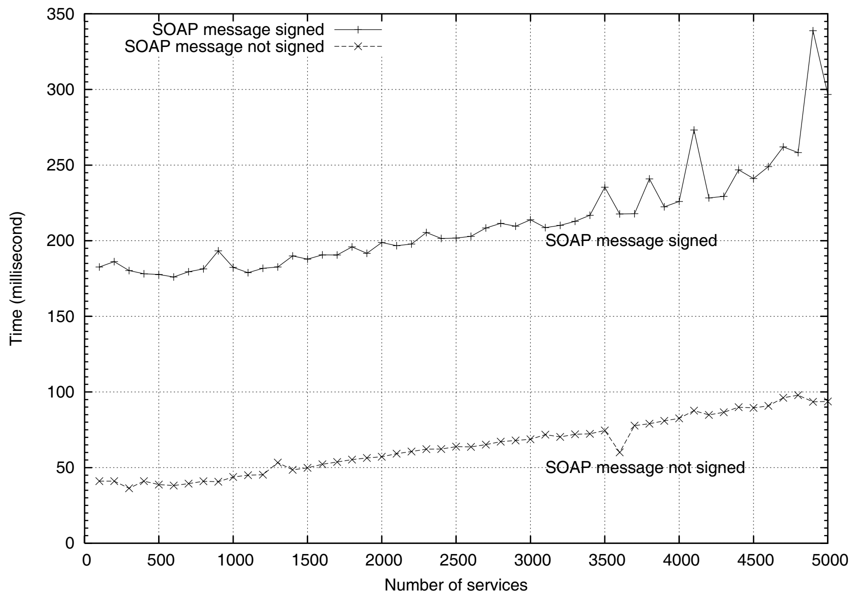

Figure 7. Inquiry overhead (WS-Security).

Table II. The time (in milliseconds) to publish and inquire for a service when there are 2000 services in the registry.

	Publish	Inquire
Signed	305.2	198.8
Not signed	82.6	57.2

5.4. GRIMOIRES versus jUDDI

jUDDI [9] is an open source standard UDDI registry, but it does not offer specific support for metadata annotation and metadata-based discovery. By comparing the performance of GRIMOIRES and jUDDI, we are able to study the overhead of representing published information in the form of RDF triples and supporting metadata-based discovery. Figures 8 and 9 compare the performance of publication and inquiry of GRIMOIRES with that of jUDDI against registry data size, respectively. Table III shows the time to publish and inquire for a service when there are 2000 services in the registry for both jUDDI and GRIMOIRES. In this test, we compare the best performant configuration of GRIMOIRES, i.e. NBIM, with jUDDI using a PostgreSQL DB backend.

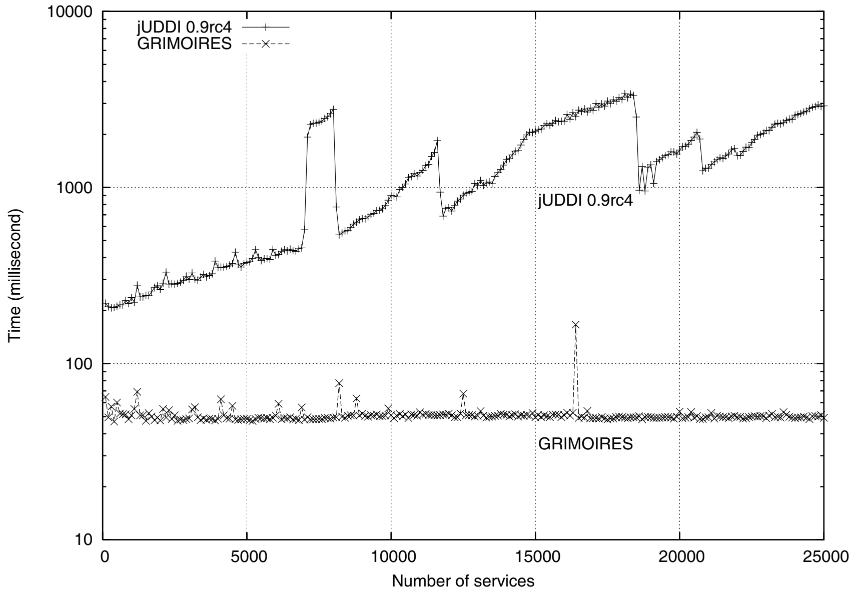

Figure 8. Publication: GRIMOIRES versus jUDDI.

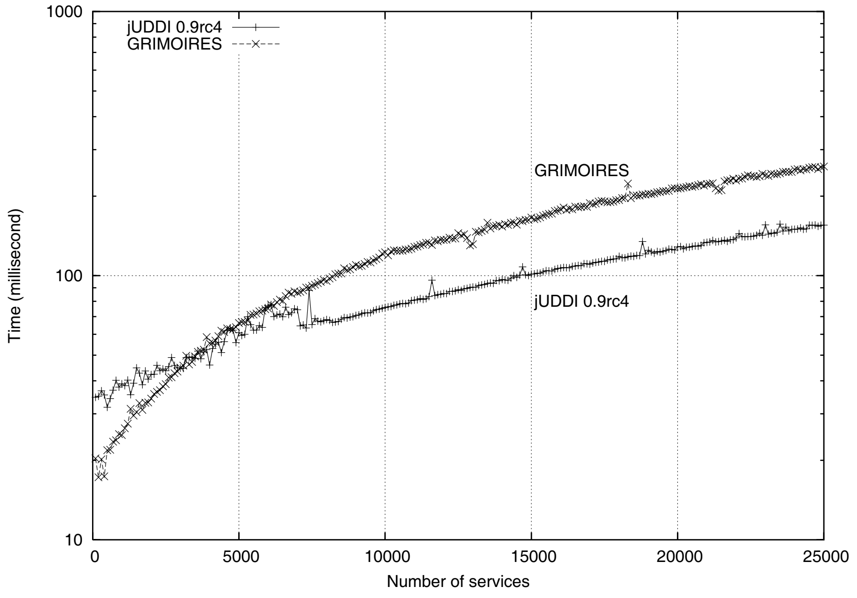

Figure 9. Inquiry: GRIMOIRES versus jUDDI.

Table III. The time (in milliseconds) to publish and inquire for a service when there are 2000 and 20 000 (as in the parentheses) services in the registry.

	Publish	Inquire
GRIMOIRES	47.55 (53.38)	34.24 (213.72)
jUDDI	263.89 (1634.01)	42.26 (128.49)

The publication overhead of GRIMOIRES is better than that of jUDDI. Our interpretation of this difference is that a native store is optimized for making RDF triples persistent, whereas in the DB, information has to be stored in multiple tables. In addition, data in NBIM may be cached by the OS which may improve the publication performance by aggregating small writes. The inquiry overhead of GRIMOIRES is 18.9% less than that of jUDDI when 2000 services are published. When the registry contains more information, the inquiry overhead of GRIMOIRES lags behind that of jUDDI. For instance, the inquiry overhead of GRIMOIRES is 66.3% higher than that of jUDDI when 20 000 services are published. The RDF triple store that introduces some penalty cost, however, offers GRIMOIRES capabilities that are not supported by jUDDI, such as discovering entities by their annotated metadata. Compared with jUDDI, GRIMOIRES offers a more specialized and targeted query capability that offsets the overhead of triple store (see Section 5.5).

In addition, the direct comparison between jUDDI and GRIMOIRES, both using a PostgreSQL backend, can be inferred by combining the performance data presented in Section 5.1 and this section. The result is not surprising: GRIMOIRES is much slower than jUDDI in such a case. However, this is due to the implementation of RDF triple store over PostgreSQL instead of the implementation of GRIMOIRES itself. Overall, this comparison shows that triple-store-based GRIMOIRES has the potential to compete with PostgreSQL based jUDDI. Compared to relational databases, triple stores are a relatively new technology; new implementations of triple stores are likely to improve their scalability.

5.5. Metadata annotation and service discovery by metadata

In this section, we study the performance of metadata annotation and service discovery by metadata, which are GRIMOIRES's extension to the UDDI specification. During the experiment, we follow the methodology presented in Section 4, except that one piece of metadata is attached to each service during the publication phase and services are discovered according to their attached metadata during the discovery phase. All the metadata are of the same annotation type, but with different values. In this test, GRIMOIRES is configured to use the FBIM store with a synchronization delay of 200 ms.

Figure 10 demonstrates the performance of metadata annotation that includes attaching metadata, deleting metadata and updating metadata, and service discovery by metadata. Table IV shows the overheads of metadata-related operations when there are 2000 services in the registry. As seen from the figure, it generally takes less than 10 ms to attach, delete, or update a piece of metadata, and the cost is almost independent of the registry data size. The 'Query by metadata' curve in Figure 10 shows the performance of discovering a service by its attached metadata. Like discovering a service by its name,

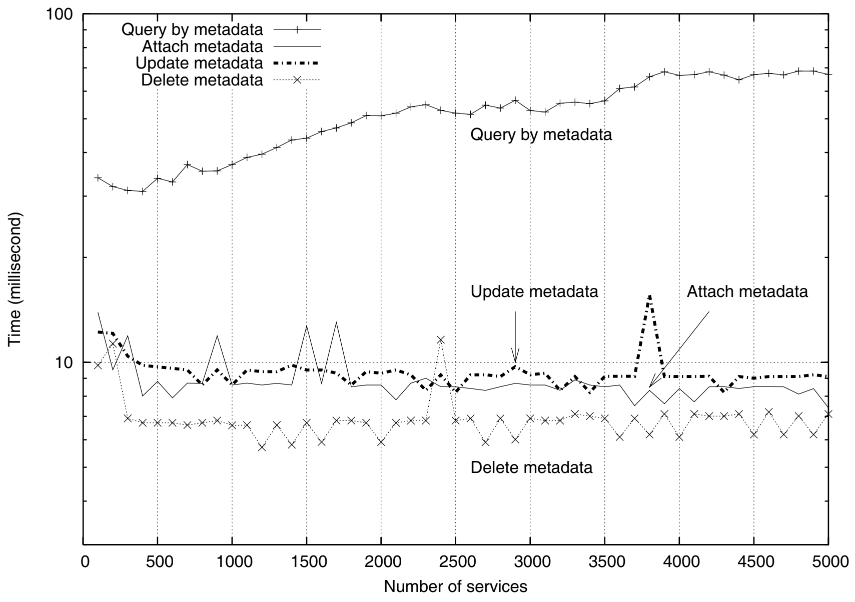

Figure 10. Metadata annotation and discovery by metadata.

Table IV. The overheads (in milliseconds) of metadata-related operations when there are 2000 services in the registry.

Attach metadata	8.6
Update metadata	9.3
Delete metadata	5.9
Query by metadata	51.0

the overhead of which is measured in Section 5.4, the overhead of discovering a service by its attached metadata is also proportional to the registry data size.

5.6. Throughput

A service registry, like any server, is designed to have a reasonable performance under heavy workload. In particular, it should respond well towards massive simultaneous requests, i.e. avoiding significant performance degrading when the number of simultaneous requests dramatically increases. The throughput benchmark evaluates a server's performance under heavy workload, by telling how

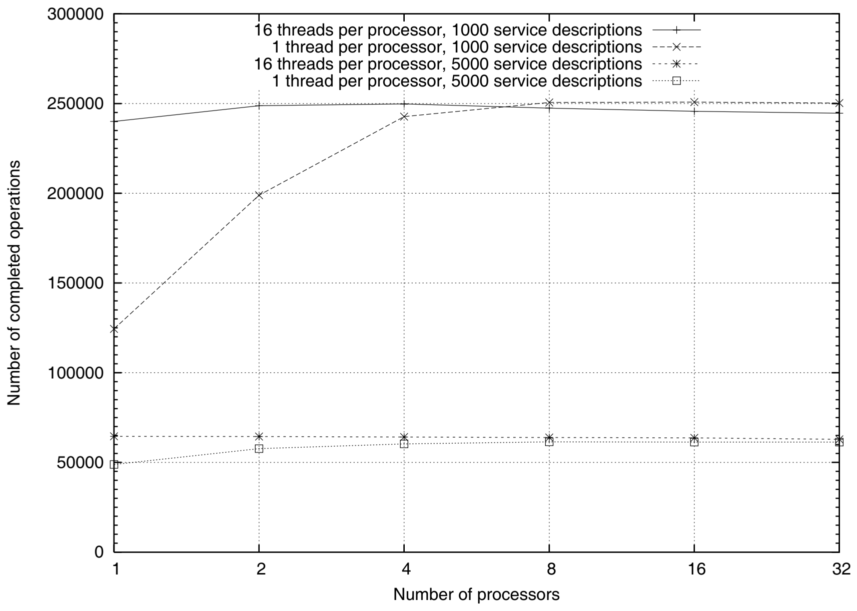

Figure 11. Throughput in 30 min.

many requests can be handled by the server within a period of time. In this test, GRIMOIRES is configured to use an in-memory store.

Figure 11 presents GRIMOIRES's throughput under massive simultaneous requests that are generated by a MPI program running on a 32 node cluster, in which each process is multi-threaded. All threads repeatedly generate a new request as soon as the previous one is replied. Since it is expected that service discovery is the most common operation acting on a service registry, the performance of service discovery operation is what we are mostly concerned with. We measure GRIMOIRES's throughput with regard to the service discovery (by name) operation. The throughput is measured against two registry data sizes: when GRIMOIRES contains 1000 service descriptions and 5000 service descriptions. When 32 processors and 16 threads per processor are leveraged at the client side, the total number of completed requests reaches a plateau, without degrading. Under such a situation, GRIMOIRES can serve 62 877 service discovery requests in 30 min when there are 5000 service descriptions registered in GRIMOIRES, i.e. 34.9 per second. Each request takes 28.6 ms.

6. CONCLUSION AND FUTURE WORK

In this paper, we have studied GRIMOIRES from the performance perspective. We have analyzed the scalability of GRIMOIRES against the amount of information published. GRIMOIRES, as a UDDI compliant service registry with metadata annotation and semantic discovery extensions, offers more useful functionalities than a standard UDDI registry, and at the same time demonstrates a performance comparable with a standard UDDI registry. Based on this experimentation, we claim that GRIMOIRES is an efficient semantics-aware service discovery engine.

Currently, we are exposing UDDI service descriptions in GRIMOIRES and their metadata annotations as WS-Resources, so that they can be operated by following the WSRF specifications [14]. By doing this, we add new features to GRIMOIRES, such as allowing service descriptions being subscribed for notification on updated and allowing service descriptions being discovered by using an XML query language (e.g., XPath).

ACKNOWLEDGEMENTS

GRIMOIRES is a project of the managed programme of the Open Middleware Infrastructure Institute (OMII), funded by EPSRC Grant GR/S90843/01.

REFERENCES

1. UDDI Specification. http://www.uddi.org [9 May 2006].
2. ebXML Registry standards. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=regrep [9 May 2006].
3. OWL-S. http://www.daml.org/services/owl-s/ [9 May 2006].
4. Miles S, Papay J, Payne T, Luck M, Moreau L. Towards a protocol for the attachment of metadata to Grid service descriptions and its use in semantic discovery. *Scientific Programming* 2004; **12**(4):201–211.
5. WWW Consortium. RDF Primer. http://www.w3.org/TR/rdf-primer/ [9 May 2006].
6. Berners-Lee T, Hendler JA, Lassila O. The semantic Web. *Scientific American* 2001; **284**(5):34–43.
7. Wroe C, Goble C, Greenwood M, Lord P, Miles S, Papay J, Payne T, Moreau L. Automating experiments using semantic data on a bioinformatics Grid. *IEEE Intelligent Systems* 2004; **19**(1):48–55.
8. Oasis Web Services Security (WSS). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss [9 May 2006].
9. jUDDI 0.9rc4. http://ws.apache.org/juddi/ [9 May 2006].
10. Wilkinson MD, Links M. BioMOBY: An open-source biological Web services proposal. *Briefings in Bioinformatics* 2002; **4**(3):331–341.
11. The GLUE schema. http://infnforge.cnaf.infn.it/glueinfomodel/ [9 May 2006].
12. Jena Semantic Web Framework. http://jena.sourceforge.net/ [9 May 2006].
13. Sesame RDF database. http://www.openrdf.org [24 April 2007].
14. Foster I, Czajkowski K, Ferguson DF, Frey J, Graham S, Maguire T, Snelling D, Tuecke S. Modeling and managing state in distributed systems: The role of OGSI and WSRF. *Proceedings of the IEEE* 2005; **93**(3):604–612.