ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI INGEGENERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA BIOMEDICA

TITOLO DELL'ELABORATO

Analisi delle prestazioni della piattaforma semantica Smart-M3 mediante il benchmark LUBM

Elaborato in
Calcolatori elettronici

Relatore:
Chiar.mo Prof. Luca Roffia

Presentata da
Lorenza Bertarelli

Correlatori:
Ing . Fabio Viola
Prof. Alfredo D'Elia

Sessione II
Anno Accademico 2014/2015

INDICE

ABSTRACT

Il software Smart-M3, ereditato dal progetto europeo SOFIA, conclusosi nel 2011, permette di creare una piattaforma d'interoperabilità indipendente dal tipo di dispositivi e dal loro dominio di utilizzo e che miri a fornire un Web Semantico di informazioni condivisibili fra entità software e dispositivi, creando ambienti intelligenti e collegamenti tra il mondo reale e virtuale. Questo è un campo in continua ascesa grazie al progressivo e regolare sviluppo sia della tecnologia, nell'ambito della miniaturizzazione dei dispositivi, che delle potenzialità dei *sistemi embedded*. Questi sistemi permettono, tramite l'uso sempre maggiore di sensori e attuatori, l'elaborazione delle informazioni provenienti dall'esterno.

È evidente, come un software di tale portata, possa avere una molteplicità di applicazioni, alcune delle quali, nell'ambito della Biomedica, può esprimersi nella telemedicina e nei sistemi e-Heath. Per e-Health si intende infatti l'utilizzo di strumenti basati sulle tecnologie dell'informazione e della comunicazione, per sostenere e promuovere la prevenzione, la diagnosi, il trattamento e il monitoraggio delle malattie e la gestione della salute e dello stile di vita.

Obiettivo di questa tesi è fornire un set di dati che mirino ad ottimizzare e perfezionare i criteri nella scelta applicativa di tali strutture. Misureremo prestazioni e capacità di svolgere più o meno velocemente, precisamente ed accuratamente, un particolare compito per cui tale software è stato progettato. Ciò si costruisce sull'esecuzione di un benchmark su diverse implementazioni di Smart-M3 ed in particolare sul componente centrale denominato SIB (Semantic Information Broker).

1. Introduzione

1.1 Panorama applicativo

L'Internet of Things (IoT) è un nuovo paradigma in cui il mondo delle tecnologie dell'informazione e della comunicazione è strettamente integrato con il mondo reale delle cose [1]. L'idea di IoT si basa sulla presenza intorno a noi di una varietà di cose o di oggetti, come sensori, attuatori o telefoni cellulari interoperabili che sono in grado di interagire tra loro e cooperare con i loro vicini per raggiungere uno scopo specifico. L'Internet of Things offre numerose opportunità in diversi settori e contesti. Tra gli ambiti applicativi più consolidati in Italia troviamo oggetti che rispecchiano solo marginalmente le caratteristiche di apertura e raggiungibilità che caratterizzano l'Internet of Things come: l'antintrusione e la videosorveglianza, la gestione delle flotte aziendali, la tracciabilità di "oggetti di valore", come ad esempio apparecchiature elettrobiomedicali, la manutenzione di dispositivi e impianti, il monitoraggio del traffico cittadino e la localizzazione dei mezzi utilizzati per il trasporto pubblico. Al gruppo degli ambiti consolidati appartengono anche soluzioni caratterizzate da una maggiore raggiungibilità degli oggetti e in alcuni casi dalla presenza di funzionalità di elaborazione dati in locale, ad esempio: i contatori intelligenti per la misurazione dei consumi elettrici, le soluzioni domotiche per l'energy management, la sicurezza delle persone e la gestione di scenari ambientali e i servizi di infomobilità.

L'ecosistema che costituisce l'Internet of Things è attualmente caratterizzato da soluzioni altamente eterogenee a livello di hardware, software e protocolli di comunicazione tra gli oggetti del mondo reale e il mondo digitale di Internet. La mancanza di standard per i primi due livelli (HW, SW e comunicazione) ha dunque portato a soluzioni in cui l'aspetto chiave dell'interoperabilità è gestito solo a livello di dato, attraverso apparati e soluzioni con funzioni di gateway. Un altro aspetto chiave della futura Internet of Things sarà comunque la standardizzazione dei protocolli di comunicazione e applicativi tra i dispositivi [2].

1.2 Telemedicina e IoT

L'OMS (Organizzazione Mondiale della Sanità) adotta nel 1997 la seguente definizione:

"*La telemedicina è l'erogazione di servizi sanitari, quando la distanza è un fattore critico, per cui `è necessario usare, da parte degli operatori, le tecnologie dell'informazione e delle telecomunicazioni al fine di scambiare informazioni utili alla diagnosi, al trattamento ed alla prevenzione delle malattie e per garantire un'informazione continua agli erogatori di prestazioni sanitarie e supportare la ricerca e la valutazione della cura*." [3].

La telemedicina nasce con lo scopo di migliorare la qualità di vita del paziente, agevolare e migliorare la qualità del lavoro di medici e infermieri, incrementare l'efficienza e la produttività del servizio sanitario. Da allora un ampio numero di sistemi è stato realizzato con lo scopo principale di ridurre il tempo che il personale medico deve spendere per il paziente garantendo al contempo un alto livello di qualità. Inizialmente i sistemi di telemedicina erano dedicati a specifiche applicazioni quali la tele-cardiologia, tele-radiologia, tele-dialisi, etc. Con l'avvento di nuove piattaforme, rese possibili dalla presenza di Internet (incluse piattaforme mobili) e con lo sviluppo di Location Based Service, la collezione di dati acquisiti da differenti sensori (indossabili, stazionari, posizionati in casa piuttosto che in altri ambienti) è diventata una realtà. Quindi l'applicazione della telemedicina porta l'assistenza a casa del paziente, consente la consultazione tra specialisti lontani, favorisce la condivisione delle conoscenze e dei protocolli diagnostico-terapeutici, mettendo a disposizione dell'intero sistema assistenziale un'infrastruttura di gestione delle informazioni cliniche potente ed efficace.

Lo sviluppo di nuovi sensori e attuatori hanno aperto la strada a nuovi scenari in campo sanitario e nuove opportunità per i servizi di healthcare attualmente investigate in progetti di ricerca [4]. Ne è un esempio CHIRON, un progetto UE (2010-2012), che propone un'architettura per la gestione efficace della salute e una completa assistenza sanitaria. La sfida principale è quella di integrare in sistemi sanitari le più recenti informazioni sui pazienti con i loro dati storici, al fine di trasformare le informazioni raccolte in un valido supporto diagnostico. Prima di tutto, i dati raccolti da più fonti eterogenee, (sensori,archivi e database)

devono essere resi facilmente reperibili e per questo immagazzinati e uniformemente rappresentati in una piattaforma condivisa. A tale scopo CHIRON definisce i seguenti tre livelli: il livello utente, il piano medico, e il piano statistico. Il primo riguarda le interazioni coi pazienti (monitoraggio e feedback locale). Il piano medico mira all'interazione con il personale medico (valutazione dei dati clinici, diagnosi, pianificazione del trattamento e l'esecuzione). Ed infine il piano statistico interessa le interazioni con ricercatori. Il ciclo di cure integrato si basa sul continuo scambio di informazioni tra questi tre livelli. La piattaforma proposta è un classico esempio di interoperabilità per la gestione di un spazio informativo,consente un approccio personalizzato in assistenza sanitaria e adempie i requisiti proposti dalla telemedicina [5].

1.3 Un requisito fondamentale: l'interoperabilità

"L'interoperabilità è la capacità di due o più sistemi, reti, mezzi, applicazioni o componenti di scambiare informazioni fra loro e di essere poi in grado di utilizzarle." [6].

In una società globalizzata che vede una sempre crescente diversità di sistemi e di applicazioni, l'interoperabilità rende possibile lo sviluppo di mercati e sistemi globali, prevenendo gli indesiderati effetti della frammentazione. Dunque l'interoperabilità è la capacità di un sistema o di un prodotto informatico di cooperare e di scambiare informazioni con altri sistemi o prodotti in maniera più o meno completa e priva di errori, con affidabilità ed ottimizzazione delle risorse. L'obiettivo è facilitare l'interazione fra sistemi differenti, nonché lo scambio e l'utilizzo di informazioni anche fra sistemi informativi non omogenei. L'interoperabilità è organizzata su tre livelli [7]:

- **Tecnica**: consente ai sistemi informativi di scambiarsi dati e servizi attraverso interfacce, formati e protocolli standard.
- **Semantica**: consente di attribuire il medesimo significato alle informazioni scambiate automaticamente fra i sistemi informativi cooperanti.
- **Organizzativa**: consente di attribuire ruoli, doveri e responsabilità precise ad ogni organizzazione coinvolta in processi condivisi.

2 Web Semantico

2.1 Verso il Web Semantico

Parallelamente alla diffusione di forme di connettività continua, di interconnessione e aggregazione tra individui e oggetti, si è verificato uno sviluppo e un consolidamento di standard per l'interoperabilità, lo scambio dell'informazione, la rappresentazione di dati e metadati, l'organizzazione, la gestione e il riuso della conoscenza. In particolare, il Web semantico agisce come collante ed è il luogo in cui l'informazione è organizzata secondo modelli condivisi e diventa significativa anche per la macchina, che può trattarla in maniera efficiente ed automatica [8]. Lo scenario attuale del WWW (World Wide Web) è quello di un enorme insieme di pagine HTML collegate fra loro attraverso link ed accessibili attraverso il protocollo HTTP (Hyper Text Transfer Protocol). Gli utenti si muovono quindi sul web grazie alla loro esperienza di navigazione e alla capacità di evocazione che possono avere parole o espressioni chiave.

Il termine "Web Semantico" è stato proposto per la prima volta nel 2001 da Tim Berners Lee [9]. Da allora il termine è stato associato all'idea di un web nel quale agiscano agenti intelligenti, cioè applicazioni in grado di comprendere il significato dei testi sulla rete e perciò in grado di guidare l'utente direttamente verso l'informazione ricercata, oppure sostituirsi a lui nello svolgimento di alcune operazioni. Un agente dovrebbe:

- Comprendere il significato dei testi presenti sulla rete.
- Creare percorsi in base alle informazioni richieste dall'utente, guidandolo poi verso di esse.
- Spostarsi di sito in sito, collegando logicamente elementi diversi dell'informazione richiesta.
- Verificare l'attendibilità di un'informazione.

Con questa tecnologia è possibile automatizzare la ricerca delle pagine, poiché all'atto della creazione del contenuto le informazioni sono definite ed inserite secondo precise regole semantiche. Il Web Semantico è un'estensione del web corrente, in cui le informazioni hanno un ben preciso significato e in cui computer e utenti lavorano in cooperazione. Ciò che si vuole creare è un sistema che

automaticamente combina la conoscenza proveniente da fonti diverse e che sia in grado di crearne di nuova.

2.2 Architettura del Web Semantico

Il codice in grado di compiere operazioni semantiche dipende dallo schema utilizzato per archiviare le informazioni, ovvero da un insieme di regole sull'organizzazione dei dati. Il principio centrale del Web Semantico è l'utilizzo di metadati, necessari per annotare le risorse e per descrivere le relazioni che essi hanno con altre risorse, utilizzando formalismi, atti ad essere processati da sistemi software [10][11][12]. Si compone dei seguenti 6 livelli:

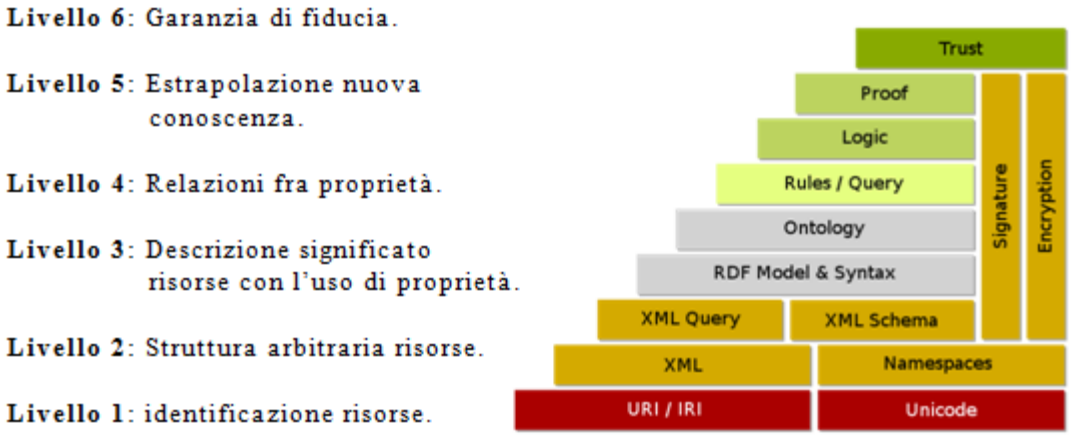


Figura 1 architettura web semantico

Il Web Semantico è un ambiente dichiarato in cui si specifica il significato dei dati e non il modo in cui si intende utilizzarli. La semantica dei dati consiste nel dare alla macchina delle informazioni utili in modo che essa possa utilizzare i dati nel modo corretto e la tecnologia di riferimento per la codifica. Lo scambio e il riutilizzo di metadati è basato sul modello con cui tutti gli elementi sono rappresentabili da triple RDF[16] sintatticamente corrette, ma prive di senso. Per attribuirvi un senso è necessario introdurre classi di oggetti iniziando a definire il dominio a cui fa rifermento l'informazione che si vuole rappresentare. Si hanno dunque:

- Dati
- metadati (riportano i dati ai concetti di uno schema)
- classi di dati (stabiliscono le relazioni fra i concetti dello schema)

Infine la rappresentazione della conoscenza e le regole che permettono di dedurre ed esportare ulteriore conoscenza è definita da una ontologia. Per la definizione delle ontologie i linguaggi maggiormente utilizzati sono RDFs e OWL [20]. Per la loro interrogazione sono stati definiti diversi linguaggi e ambienti, tra cui SPARQL (Simple Protocol and RDF Query Language) [27]. Quest'ultimo è lo standard proposto dal W3C e quello attualmente più diffuso e comprende tre diverse specifiche: il protocollo per l'accesso ai dati per l'interrogazione remota di basi di dati RDF, il linguaggio per definire le *query* e il formato per la rappresentazione dei risultati.

2.3 Livello1: standard URI

Il Web si basa sullo standard URI (Uniform Resource Identifiers)[13] per la definizione univoca di indirizzi internet. Consiste nell'attribuzione di una stringa che identifica univocamente una risorsa generica, che può essere un indirizzo web, un documento, un'immagine, un file, un servizio, ect. Gli URI costituiscono la tecnologia di base ideale con la quale costruire un web globale, in quanto è possibile con essi definire qualsiasi oggetto e poi immetterlo nel web. L'URI diviene un nome per una risorsa, accessibile o meno attraverso internet.

La produzione e l'uso di URI non è controllata da nessuna persona o organizzazione, ciò vi conferisce flessibilità, ma al contempo è causa di perdita di unicità. Per ovviare a questo problema al medesimo livello è associato all'URI un sistema unicode. Si tratta di un sistema di codifica (sequenza di byte) che assegna un numero univoco ad ogni carattere usato per la scrittura di testi, indipendentemente dalla lingua, piattaforma informatica o programma. Prevede una codifica fino a 21 bit e supporta un repertorio di codici numerici che possono rappresentare circa un milione di caratteri. Ciò appare sufficiente a coprire anche i fabbisogni di codifica di scritti del patrimonio storico dell'umanità, nelle diverse lingue e negli svariati sistemi di segni utilizzati. Si prefigge dunque di coprire tutti i caratteri rappresentabili, garantendo la compatibilità e la non sovrapposizione con le codifiche dei caratteri già definiti, ma lasciando comunque dei ben precisi campi di codici "non usati", da riservare per la gestione autonoma all'interno di applicazioni particolari.

2.4 Livello 2: XML e namespace

L'XML (eXtesible Markup Language) non ha come scopo principale quello di descrivere la formattazione e la visualizzazione di un documento, ma semplicemente quello di descrivere la struttura dell'informazione, quindi la sintassi [14]. L'XML è un framework per la definizione di linguaggi. Ogni linguaggio XML rivolto a un particolare dominio applicativo è intrinsecamente internazionale e indipendente dalla piattaforma, ma i linguaggi hanno alcune caratteristiche condivise:

- utilizzano la stessa sintassi base per il markup
- sfruttano lo stesso insieme di strumenti generici per l'elaborazione dei documenti.

Non è un linguaggio singolo che può essere esteso per adattarsi ad altri usi, ma una notazione comune con cui è possibile costruire linguaggi markup. Questa libertà tuttavia lo rende poco adatto a definire completamente la struttura e l'interscambio di informazioni tra diverse realtà. È stata quindi favorita la creazione di nuovo linguaggio: il linguaggio RDF.

Per fissare le idee prendiamo in considerazione la rappresentazione di un generico articolo a carattere tecnico e proviamo a rappresentarlo secondo il modello XML:

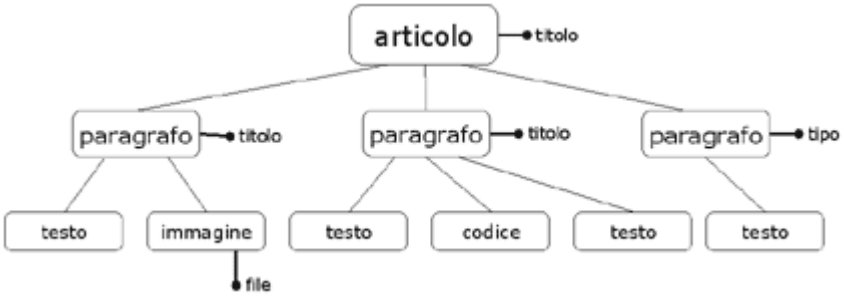


Figura 2 Esempio di albero XML

Nella figura abbiamo un elemento padre denominato articolo che contiene una lista di elementi che rappresentano i vari paragrafi dell'articolo. Ciascun paragrafo a sua volta contiene del testo, degli esempi di codice e delle immagini. La maggior parte degli elementi di questa struttura ad albero di documenti possiede degli attributi: titolo, tipo, file [15]. La struttura logica di un documento XML dipende

dalle scelte progettuali. Siamo noi a decidere come organizzare gli elementi all'interno di un documento XML. Non esistono regole universali per l'organizzazione logica di un documento se non il buon senso e l'esperienza. La struttura logica di un documento XML viene tradotta in una corrispondente struttura fisica implementata tramite un file di testo, creato con un qualsiasi editor e composta di elementi sintattici chiamati tag: questi forniscono una notazione per arricchire i documenti di testo e la possibilità di collegare altri documenti e file.

La rappresentazione fisica del documento XML visto prima può essere la seguente:

```
<?xml version="1.0" ?>
<articolo titolo="Titolo dell'articolo">
   <paragrafo titolo="Titolo del primo paragrafo">
      <testo>
         Blocco di testo del primo paragrafo
      </testo>
      <immagine file="immagine1.jpg">
      </immagine>
   </paragrafo>
   <paragrafo titolo="Titolo del secondo paragrafo">
      <testo>
         Blocco di testo del secondo paragrafo
      </testo>
      <codice>
         Esempio di codice
      </codice>
      <testo>
         Altro blocco di testo
      </testo>
   </paragrafo>
   <paragrafo tipo="bibliografia">
      <testo>
         Riferimento ad un articolo
      </testo>
   </paragrafo>
</articolo>
```

Figura 3 esempio di codice XML

2.5 Livello 3: RDF

RDF (Resource Description Framework) è uno standard proposto dalla W3C come set di linguaggi dichiarativi, adattato a descrivere la struttura di una parte della realtà. XML è invece il formato di serializzazione più usato per RDF. L'RDF

è lo strumento base per la codifica, lo scambio e il riutilizzo di metadati strutturati e consente interoperabilità tra applicazioni che si scambiano informazioni sul web.

Non descrive la semantica, ma fornisce una base comune per poterla esprimere definendo la semantica dei tag XML [16][17][18]. RDF è costituito da due componenti:

- RDF Model and Syntax che definisce un modello di dati RDF e la sua codifica XML senza definire relazioni.
- RDF Schema che definisce specifici vocabolari per i metadati e creare nessi fra gli oggetti.

L'RDF-Model fornisce un modello per descrivere le risorse che possono avere delle proprietà. Definiamo come **risorsa** qualunque cosa che può essere rappresentata da un URI, come **proprietà** una specifica caratteristica o attributo di una risorsa e come **asserzione** la tripla composta da una risorsa, una proprietà e un valore o un URI (in quest'ultimo caso questo punta ad un'altra risorsa per quella proprietà). L'asserzione descrive dunque le caratteristiche di una risorsa e le relazioni con altre risorse e la possiamo rappresentare come:

SOGGETTO+PREDICATO+OGGETTO.

L'RDF-Schema definisce dei vocabolari, quindi l'insieme delle proprietà semantiche.Questo permette di definire nuovi tipi di classi e gerarchie di classi. Deriva dai Namespace XML e fornisce un metodo per identificare in maniera non ambigua la semantica e le convenzioni dell'uso delle proprietà. Nella seguente figura è raffigurato un modo per esprimere lo statement RDF in forma grafica utilizzando un grafo etichettato orientato nel quale la risorsa è rappresentata da un'ellisse, la proprietà da un arco orientato, che parte dalla risorsa e punta all'oggetto, e l'oggetto da un rettangolo.

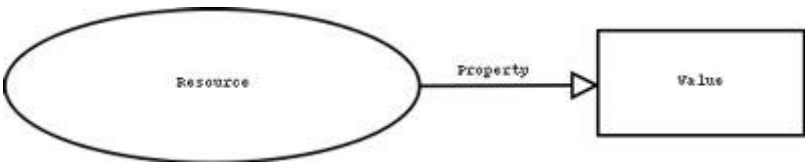


Figura 4 rappresentazione soggetto-predicato-oggetto

Per cercare di comprendere meglio il data model RDF consideriamo un esempio in cui andiamo a descrivere, mediante RDF, il metadato riguardante l'autore di una pagina HTML. L'informazione che vogliamo descrivere è la seguente: Mario Rossi è l'autore della pagina http://nome_di_un_dominio/esempio.html

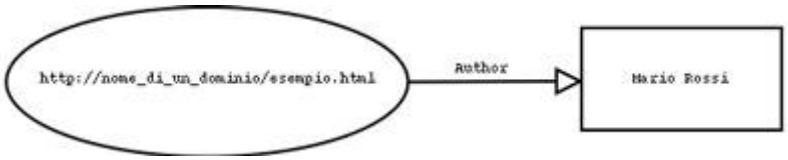


Figura 5 prima rappresentazione soggetto-predicato-oggetto attraverso il modello RDF

L'oggetto di uno statement RDF può essere a sua volta una risorsa, consentendo in questo modo di descrivere in maniera più approfondita il valore della proprietà. Consideriamo l'esempio precedente e vediamo di aggiungere maggiori informazioni riguardanti l'autore della pagina HTML (ad esempio e-mail e numero di telefono). In questo caso dobbiamo aggiungere allo statement precedente, la descrizione della risorsa autore che può essere identificata in maniera univoca utilizzando ad esempio l'URI della sua Homepage. Lo statement RDF espresso in forma grafica diventa quindi:

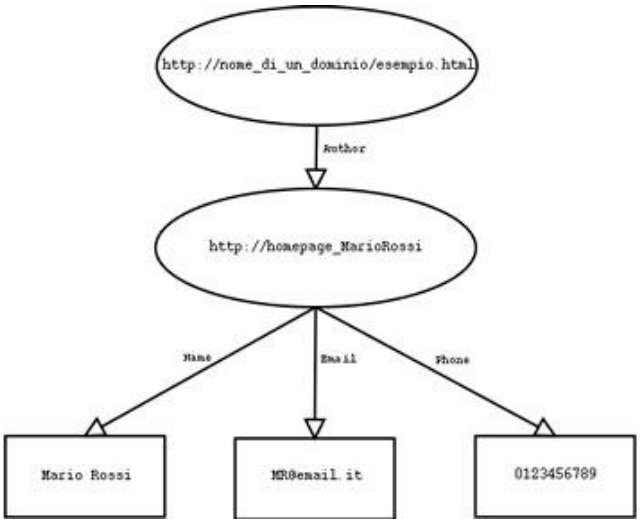


Figura 6 seconda rappresentazione soggetto-predicato-oggetto attraverso il modello RDF

Il primo esempio di statement RDF può essere espresso, utilizzando la sintassi XML, nel seguente modo:

```
<?xml version=”1.0”?>
<rdf:RDF
  xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
  xmlns:a=”http://nome_di_un_dominio/schema_autore/”>
  <rdf:Description about=”http://nome_di_un_dominio/esempio.html”>
    <a:author>
      Mario Rossi
    </a:author>
  </rdf:Description>
</rdf:RDF>
```

Figura 7 prima rappresentazione riscritta con una sintassi XML

L’elemento <rdf:RDF> racchiude la definizione dello statement RDF ed al suo interno troviamo la definizione di due Namespace: il primo è relativo al Namespace RDF, mentre il secondo Namespace contiene l’URI che identifica lo schema RDF utilizzato per descrivere la semantica e le convenzioni che regolano l’utilizzo delle proprietà presenti nello statement. La descrizione del metadato è contenuta all’interno dell’elemento <rdf:Description> ed il suo attributo *about* identifica la risorsa alla quale si riferisce il metadato stesso. La proprietà dello statement è descritta utilizzando il tag <a:author>, secondo le regole che sono espresse nel relativo schema RDF. Il secondo esempio di statement RDF, espresso secondo la sintassi XML, diventa:

```
<?xml version=”1.0”?>
<rdf:RDF
  xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
  xmlns:a=”http://nome_di_un_dominio/schema_autore/”>
  <rdf:Description about=”http://nome_di_un_dominio/esempio.html”>
    <a:author rdf:resource=”http://homepage_MarioRossi/”/>
  </rdf:Description>

  <rdf:Description about=”http://homepage_MarioRossi/”>
    <a:name>Mario Rossi</a:name>
    <a:email>MR@email.it</a:email>
    <a:phone>0123456789</a:phone>
  </rdf:Description>
</rdf:RDF>
```

Figura 8 seconda rappresentazione riscritta con una sintassi XML

In questo caso, all’interno dello statement RDF abbiamo la definizione di due risorse (identificate dai due elementi <rdf:Description>) che sono messe in relazione attraverso l’uso dell’attributo *rdf:resource* presente nell’elemento

<a:author> [19]. In questo modo la descrizione della seconda risorsa (quella relativa all'autore del documento) viene assegnata come valore della proprietà *author* della prima risorsa.

2.6 Livello 4: Ontologia

Gli schemi RDF hanno delle limitazioni, non possono infatti riconoscere le classi equivalenti e limitare il numero di proprietà da usare. Per colmare queste lacune sono state introdotte le ontologie OWL (Web Ontology Language) ovvero un insieme di regole di inferenze. Più semplicemente un'ontologia permette di specificare in modo aperto e significativo i concetti e le relazioni che caratterizzano un certo dominio di conoscenza. Il linguaggio OWL estende RDFs permettendo una notevole ricchezza semantica. Una caratteristica fondamentale è infatti la possibilità di descrivere classi in modo più ricco, distinguendo fra classi disgiunte per cui nessuna istanza può appartenere ad entrambe le classi, oppure classi come intersezione di altre due o più.

Anche per le proprietà è possibile dare descrizioni molto ricche e raffinate, potendo distinguere fra proprietà transitive, simmetriche, funzionali o come inverse di altre.

Infine, un'evoluzione importante è la possibilità di distinguere tra proprietà che hanno come valori dati semplici (DatatypeProperty) o vere e proprie classi (ObjectProperty). L'ontologia è costituita da una Taxonomy (Vocabulary + Structure) e delle Relationships, contraints and rules [20][21].

2.7 Livelli 5 e 6 : Logica, prova e fiducia

Questa parte del *Web Semantico* non è stata ancora sviluppata. L'obiettivo è quello di realizzare dei sistemi in grado di formulare ogni principio logico e permettere ai computer di ragionare, per inferenza, usando questi principi. Una volta costruiti questi sistemi "logici", potremmo utilizzarli per provare qualcosa. Persone in tutto il mondo possono scrivere istruzioni logiche. Quindi la macchina può seguire questi link "semantici" per costruire dimostrazioni [10]. In sintesi avremo:

- Un livello logico: affinché il Web Semantico possa effettivamente aiutarci in una vasta gamma di situazioni, estraendo autonomamente informazioni utili dalla mole di documenti annotati semanticamente, è indispensabile costruire un potente linguaggio logico per realizzare le inferenze (ovvero procedimenti deduttivi mediante cui, a partire da una o più premesse, si ricava, per via logica, una conclusione).
- Un livello di Prova: dove le conclusioni ottenute saranno validate tramite motori di validazione costituiti da sequenze di formule derivate da assiomi.
- Un livello Trust: con cui il sistema restituirà solo quelle informazioni che secondo il richiedente proverranno da utenti di indubbia attendibilità. Lo scenario che si vuole realizzare è che un utente comunica al suo computer che ha fiducia in una persona, a sua volta questa persona ha fiducia in altre persone e queste ultime in altre. Tutte queste relazioni di fiducia si aprono a ventaglio e formano il *Web of Trust*.

3 SMART –M3

3.1 Introduzione

La piattaforma Smart-M3 è stata sviluppata all'interno del progetto Europeo SOFIA (Smart Object For Intelligent Application), un progetto Artemis della durata di tre anni che coinvolge diciannove partner di quattro nazioni europee diverse.Obiettivo di SOFIA è stato realizzare una piattaforma innovativa di interoperabilità semantica [22][23].Smart-M3 (dove M3 sta per Multi vendor, Multi device e Multi domain) è il nome di un progetto software open-source che mira a fornire una piattaforma di scambio delle informazioni tramite la quale entità eterogenee possano interoperare. La comprensibilità delle informazioni si basa sul modello ontologico di dati che permette di creare una piattaforma di interoperabilità indipendente dal tipo di dispositivo e dal dominio di utilizzo. Prevede un metodo di memorizzazione dei dati basato sul modello RDF e utilizza un protocollo di comunicazione in formato RDF/XML che supporta il linguaggio SPARQL attraverso il quale è possibile interrogare e modificare la conoscenza.

3.2 Struttura Smart-M3

La piattaforma Smart M3 si suddivide in due parti:

- **SIB** (Semantic Information Broker), un nucleo centrale, ospitato da un dispositivo fisico, che contiene l'informazione e fornisce la base della conoscenza. Ne possono esistere diverse implementazioni, ne sono un esempio la RedSIB, monolitica e scritta in C o la SIB in tecnologia OSGi basata sul linguaggio Java e quindi avente maggiore versatilità, modularità e portabilità.
- **KP** (Knoledge Processor): qualsiasi dispositivo in grado di ospitare il software per inserire o recuperare informazioni attraverso la SIB. Può essere paragonato ad un interprete che traduce dal linguaggio specifico di un dispositivo al linguaggio parlato dalla SIB. Un KP può essere un KP cosiddetto Consumer se viene utilizzato per leggere informazioni dalla SIB ma non inviarne; viceversa può essere un KP Producer, o infine un Aggregator se svolge entrambe le funzioni. I KP possono essere scritti in un buon numero di linguaggi di programmazione attraverso una serie di API [24].

La nozione di applicazione in uno Smart Space differisce radicalmente dal concetto di applicazione tradizionale.

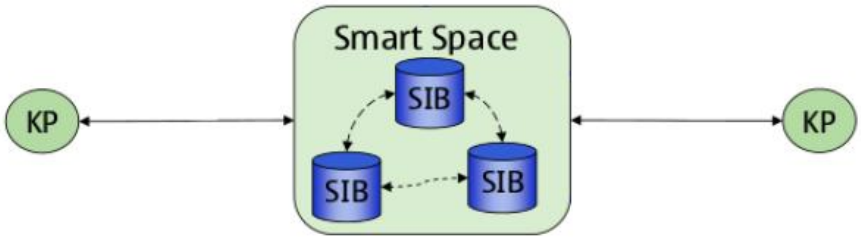


Figura 9 raffigurazione SIB e KP

Nel caso più semplice, tutta l'informazione di uno Smart Space è immagazzinata da una sola SIB, ma vi è la possibilità concettuale di connettere più SIB tra di loro. Ogni KP vede tutto il contenuto informativo dello Smart Space indipendentemente dalla SIB a cui effettivamente si connette. L'informazione in uno Smart Space viene memorizzata come grafo RDF, in accordo ad una ontologia definita ed il ruolo dei KP è di interrogare tale sistema richiedendo informazioni o

effettuando modifiche su di esse utilizzando le primitive previste dal protocollo di comunicazione con la SIB.

La comunicazione con la SIB richiede un protocollo ben definito: lo Smart Space Access Protocol (SSAP), basato su messaggi in linguaggio XML. Qualsiasi messaggio inviato alla SIB è seguito da un messaggio di risposta dalla SIB.

L'SSAP al momento prevede i seguenti tipi di messaggi:

- JOIN: è il primo messaggio da inviare alla SIB per iniziare la comunicazione e contiene le informazioni di chi effettua la richiesta di connessione. Il messaggio di risposta è l'insieme delle informazioni sulla SIB associato al rifiuto della connessione o la conferma.
- LEAVE: chiude la connessione con la SIB. Per riaprirla si deve inviare nuovamente il messaggio di Join. La risposta dalla SIB conferma o meno la chiusura della connessione.
- INSERT: è il messaggio che permette di inserire triple nella SIB. La tripla è in formato RDF ed è composta da Soggetto, Predicato e Oggetto. La SIB conferma o meno l'inserimento, ma non effettua alcun controllo sulla coerenza dei dati.
- REMOVE: è l'operazione opposta alla Insert, permette di rimuovere una lista di tripla. Anche in questo caso la SIB conferma o meno la rimozione, ma non effettua alcun controllo.
- UPDATE: ha come parametri due liste di triple. La prima viene rimossa e la seconda viene aggiunta ed ancora una volta la SIB conferma o meno le operazioni. È importante notare che quello che si ottiene con l'Update si può ottenere anche con una Remove e una Insert.
- QUERY: è il messaggio per leggere informazioni dalla SIB. Servono per cercare triple nel medesimo grafo e alle quali la SIB risponde con tutte le triple trovate.
- SUBSCRIBE: serve per ricevere una notifica dalla SIB quando viene manipolata una tripla che corrisponde al pattern specificato. La SIB conferma o meno la Subscribe e invia una notifica appena viene effettuata una Insert, Remove o Update che riguarda quella tripla.
- UNSUSCRIBE: cancella la sottoscrizione, interrompendo le notifiche.

SUBSCRIBE, UNSUSCRIBE e PUBLISH implementano il paradigma di messaggistica Publish/Subscribe dove mittenti (publishers) non sono programmati per inviare i loro messaggi a destinatari specifici (subscribers).

I messaggi sono differenziati in canali, senza alcuna informazione sulla conoscenza intrinseca. I Subscribers esprimono l'interesse per uno o più canali, ricevendo messaggi, ma senza conoscere i publishers coinvolti. Questo disaccoppiamento fra subscribers e publishers consente una maggiore scalabilità e una topologia di rete più dinamica. Dunque i client esplicitano in una SUBSCRIBE i nomi dei canali a cui sottoscriversi. I messaggi inviati da altri client per questi canali saranno quindi indirizzati dalla SIB a tutti i client iscritti. Un client sottoscritto a uno o più canali non dovrebbe eseguire comandi, se non per inscriversi o cancellarsi da o verso altri canali[25].

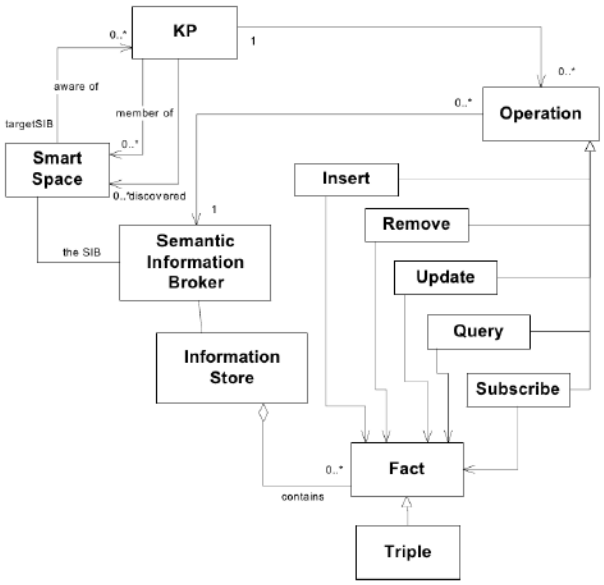


Figura 10 rappresentazione grafica dei messaggi SSAP

Vediamo come esempio due tipi di messaggi SSAP di *query* SPARQL:

- il messaggio di richiesta di esecuzione della *query* che il KP invia alla SIB
- il messaggio di risposta che contiene i risultati che la SIB invia al KP.

Per primo di seguito viene mostrata la struttura di un tipico messaggio XML del protocollo SSAP di richiesta di *query* di tipo SPARQL, che il KP invia alla SIB.

```
<SSAP_message>
   <node_id>ID</node_id>
   <space_id>ID</space_id>
   <transaction_id>INTEGER</transaction_id>
   <transaction_type>QUERY</transaction_type>
   <message_type>REQUEST</message_type>
   <parameter name = "type">
      sparql
   </parameter>
   <parameter name = "query">
      QUERY_STRING
   </parameter>
</SSAP_message>
```

Figura 11 messaggio di richiesta di esecuzione della query che il KP invia alla SIB

Come si può notare, la radice del messaggio, come per tutti i messaggi delle altre operazioni previste dal protocollo, è SSAP-message. Al suo interno contiene i seguenti tag:

- node_id, campo che identifica il KP che richiede l'operazione;
- space_id, identifica lo *smart space*;
- transaction_id, contiene il numero associato all'attuale operazione del KP;
- transaction_type, identifica il tipo di operazione da effettuare; message_type, indica se il messaggio è inviato dal KP alla SIB (REQUEST) o viceversa (CONFIRM). Inoltre, un ulteriore tipo è (INDICATION) che viene inviato dalla SIB quando deve mandare notifiche per sottoscrizioni.

Questi sono i *tag* presenti in tutti i messaggi SSAP per qualsiasi primitiva. Invece, quelli specifici per l'operazione di *query* sono:

- parameter name=”type”, campo che indica se l'operazione di *query* è del tipo SPARQL oppure RDF-M3. Queste sono gli unici due tipi di linguaggi di *query* supportati dalla SIB utilizzata;
- parameter name=”query”, campo in cui viene inserita la stringa corrispondente alla *query* SPARQL.

Di seguito è mostrata la struttura di un tipico messaggio XML del protocollo SSAP di risposta ad una richiesta di *query*, che la SIB invia al KP.

```
<SSAP_message>
   <node_id>ID</node_id>
   <space_id>ID</space_id>
   <transaction_id>INTEGER</transaction_id>
   <transaction_type>QUERY</transaction_type>
   <message_type>CONFIRM</message_type>
   <parameter name = "status">
            STATUS_CODE
   </parameter>
   <parameter name = "results">
     ...
   </parameter>
</SSAP_message>
```

Figura 12 messaggio di risposta che contiene i risultati che la SIB invia al KP

Rispetto al messaggio di richiesta *query* analizzato precedentemente, il message_type è di tipo CONFIRM ed inoltre sono presenti i seguenti due campi:

- parameter name=”status”, il quale indica lo stato della risposta, ovvero l'avvenuta o non avvenuta esecuzione della *query* da parte della SIB;
- parameter name=”results”, *tag* al cui interno è presente la risposta RDF/XML, nel caso in cui la *query* sia stata eseguita correttamente. [26]

3.3 SPARQL

SPARQL (Simple Protocol and RDF Query Language) è un linguaggio di interrogazione per RDF. Nasce infatti per manipolare e recuperare dati RDF.

Le query SPARQL si basano sul meccanismo del "pattern matching" e in particolare su un costrutto, il "triple pattern", che ricalca la configurazione a triple delle asserzioni RDF fornendo un modello flessibile per la ricerca di corrispondenze[27] [28][29][30].

```
?titolo cd:autore ?autore.
```

Figura 13 esempio di triple pattern

Questo "triple pattern" prevede due variabili (contrassegnate con ?), che fungono in un certo senso da incognite dell'interrogazione, *cd:autore* funge invece da costante: le triple RDF che trovano riscontro nella base di dati

associeranno i propri termini alle variabili corrispondenti. La struttura generica di una query è composta da:

- PREFIX: assegna un prefisso ad un namespace al fine di abbreviare gli URI. Tale prefisso può essere utilizzato per riferire tramite un Qname compatto (es. “ cd: ”) le risorse presenti nel grafo da interrogare.
- Un costrutto per indicare il tipo di query, ovvero per identificare quale informazione viene restituita. Esistono diverse forme di query in SPARQL introdotte da diverse parole chiave come:
  SELECT: fornisce l’elenco delle variabili da riportare nel risultato.
  CONSTRUCT: ottiene il grafo RDF specificato da un graph template.
  ASK: testa la presenza o assenza di una soluzione nel graph pattern.
  DESCRIBE: ottiene un grafo RDF contente dati riguardanti le risorse.
- FROM: indica la sorgente dei dati da interrogare.
- Graph pattern: set di triple pattern scritte come una sequenza. La clausola WHERE: definisce un criterio di selezione specificando tra parentesi { una o più triple, isolando un sottografo del grafo RDF.

Per chiarire ecco una semplice query di selezione SPARQL:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM  <http://cd.com/listacd.ttl>
WHERE {?titolo cd:autore ?autore.
            ?titolo cd:anno ?ann .
}
```

Figura 14 esempio di query SPARQL

Applicando la query al set di triple, si ottiene il seguente risultato:

titolo	autore	anno
"Amber"	"Autechre"	1994

Figura 15 esempio di risposat tabulare SPARQL

Nel linguaggio SPARQL possono essere rappresentati i seguenti oggetti:

- URI, rappresentabili anche tramite QNames (Qualified Names), ovvero con sintassi : <prefisso>:<ResourceName> (*cd:autor*)
- Literals

- Variabili (precedute, indifferentemente, da un ? o un $)
- Operatori su grafi

Sintassi Triple Pattern:

- I triple pattern sono separate da punti (.)
- Predicate-Object Lists: In un pattern è possibile specificare più triple aventi uno stesso soggetto indicando questo una volta sola e facendolo seguire da una lista di coppie predicato-oggetto seguite dal simbolo “;” (es. {?person foaf:knows :mario_rossi ; foaf:name ?name}).
- Object Lists: Se dei triple-pattern condividono sia il predicato che il soggetto, allora è possibile esprimere tali triple indicandone il soggetto e il predicato e separando i vari oggetti con il simbolo “,”.
- (es. { ?person foaf:knows :mario_rossi , :gino_bianchi }).

È possibile vincolare il matching del Graph Pattern utilizzando diversi operatori, funzioni o anche delle estensioni dedicate come:

- FILTER: usato per porre delle restrizioni sui valori da associare alle variabili.
- OPTIONAL: introduce una condizione opzionale, se questa non è verificata non ritorna nessun risultato, ma neanche elimina il resto delle soluzioni.
- UNION: seleziona tutte le soluzioni che sono in collegamento con almeno un triple pattern.

O modificatori di sequenza SPARQL come:

- DISINCT: permette di escludere i valori duplicati.
- LIMIT: pone restrizioni al numero di restrizioni.
- OFFSET: permette di saltare un certo numero di risultati.
- ORDER BY: imposta l’ordine dei risultati della query

4 BENCHMARK

4.1 Cos'è il benchmark

Nell'informatica il benchmark è un metodo di misurazione delle performance. Il benchmark (o *benchmarking*) consiste in un test finalizzato ad ottenere le prestazioni di esecuzione di un computer o di un programma informatico (software). Possiamo distingue due tipologie di benchmark:

- Benchmark hardware. È un test per misurare la velocità di calcolo di un computer e l'occupazione delle risorse (in memoria). Consiste nell'esecuzione di un algoritmo su diversi computer per ottenere un indicatore della prestazione in esecuzione.
- Benchmark software. È un test per misurare la velocità di esecuzione delle operazioni da parte di un apposito software. Consente di verificare il grado di efficienza di un programma informatico (software) scritto in un determinato linguaggio e tradotto in linguaggio macchina con un compilatore.

4.2 Architettura generale

Nel nostro campo applicativo facciamo riferimento ad un benchmark nato per testare le prestazioni di RDF Store, confrontandoli con altri RDF Store o ulteriori sistemi di performance, come SPARQL to SQL rewriter o database relazionali. Inoltre esistono numerose pubblicazioni che si incentrano sulla valutazione del carico, del ragionamento e delle prestazioni delle query, mentre pochi sono gli studi effettuati per la valutazione delle performance dei sistemi pubblish/subscribe. Nello specifico ci siamo basati sul benchmark LUBM: A benchmark for OWL knowledge base systems.

L'obiettivo che ci siamo proposti di raggiunger con questa tesi è la valutazione della SIB, dunque non applicheremo tale benchmark all'RDF Store, ma al sistema che si interfaccia ad esso e fa da tramite nella comunicazione con i KP. Abbiamo effettuato i test sul server "mml.arces.unibo.it" sul quale sono state rese disponibili le seguenti SIB:

– SIB OSGi non persistente;

- SIB OSGi persistente;
- RedSIB non persistente _ con storage in RAM basato su Berkeley DB;
- RedSIB persistente _ BerkeleyDB 5.3;
- RedSIB persistente _ Virtuoso;
- RedSIB non persistente _ con storage in RAM basato su hash.
- CuteSIB non persistente

4.3 Suite di programmi utilizzati

È stato realizzato come primo componente il programma per le query "query_test". Tale programma ha le seguenti caratteristiche: è scritto in Python, è in grado di testare più SIB in modo sequenziale, accetta in ingresso query SPARQL ed è in grado di iterare le query un numero prefissato di volte. Supporta anche l'inserimento della SIB leggendolo da file in formato N3. Questo consente di sfruttare lo stesso programma anche per altri benchmark tra i quali ad esempio SP2B [31] e non soltanto per LUBM. Infine fornisce in uscita un file CVS (Comma Separated Value) contenente tutte le tempistiche, le medie e il numero di risultati riportati e un grafico in formato SVG (Scalable Vector Graphic), ovvero un linguaggio di grafica vettoriale bidimensionale basato su XML sviluppato dal consorzio W3C. Lo sviluppo della suite di test è coadiuvato da un sistema di revisione (Git), il cui repository è su Github (un servizio web di hosting per lo sviluppo di progetti software, che può essere utilizzato anche per la condivisione e la modifica di file di testo) all'indirizzo:

https://github.com/desmovalvo/SmartM3_testsuite.git.

Mentre il primo programma è utilizzato per valutare le query, il secondo è adoperato per valutare i tempi di inserimento della conoscenza di base della SIB.

Questo è , "Filler", anche'esso scritto in Phyton con una licenza GPL, consente di caricare il dataset di dati sulla SIB in formato OWL e N3 con uno step di 100 ontologie per volta, fornendo in uscita il numero di triple che devono essere inserite, quelle effettivamente inserite e i tempi di caricamento.

Per comprendere appieno quanto eseguito dalle singole query di test, ci siamo serviti di un set di programmi e applicazioni utili all'analisi. Questi sono:

- L'applicazione SmartDashboard versione 1.0.0.0. associata a SEPA SPARQL Dashboard versione 1.0.5682.19155. La prima consente di caricare file .owl prima sulla cache poi sulla SIB e di visualizzare graficamente classi e proprietà di tali ontologie. La seconda invece, fornisce una interfaccia per definire i parametri di connessione (IP del server e porta della SIB) e verificare la connessione tramite ping (online). Effettuato il collegamento l'applicazione consente di eseguire un set di query o update specificando il tipo di cliente come produttore o consumatore. Queste, attuate singolarmente o caricate da un file .xml, sono raffigurate negli appositi spazi. Una volta effettuate è possibile verificare la loro riuscita e i tempi (ms) di esecuzione, mentre i risultati sono visualizzati sotto forma di triple e quindi facilmente consultabili.
- Protégé. Un editor open-source di ontologie gratuito, che fornisce un'interfaccia utente grafica, per definire ontologie o come nel nostro caso la possibilità di estrapolare le informazioni di base sulle entità come: Class Hiearchy, TopDataProperty, ObjectProperty, Domains e Ranges utili per la corretta comprensione delle query.

4.4 SIB valutate

La SIB, come già menzionato, è la responsabile della conversione e controllo delle applicazioni condivise in Smart-M3. Sono state implementate diverse SIB, che verranno valutate e confrontate in questa tesi.

4.4.1 RedSIB

Nello specifico la RedSIB è basata su uno store Redland (progettato all'inizio del 2000), un insieme di librerie di software, che forniscono il supporto per l'RDF.

RedSIB è scritta in C e fa uso delle librerie D-Bus (il che ne limita l'utilizzo al solo sistema operativo GNU/Linux). Il Redland Context ha il compito comune di fondere grafi e monitorare le sorgenti. In particolare si occupa di abilitare la fusione/aggiornamento/scissione di grafi, identificare sottografi con nodi-context,

dichiarare identità, dichiarare provenienze e aggiungere triple attraverso un nodo-context diverso, che può essere oggetto di altre dichiarazioni [32][33]. Redland presenta [34]:

- librerie di base e API (Application Programming Interface) per manipolare il grafico RDF.
- Storage per i grafici.
- Interrogazioni con SPARQL.
- Più linguaggi per leggere e scrivere RDF come RDF/XML, N-triple e turtle.
- La possibilità di aggregare dati.

L'API degli utenti prevede che le triple possano essere aggiunte, rimosse o cercate facilmente. Per la memorizzazione delle triple esistono due classi di store:

- "in-memory store" nel quale viene selezionato un nodo come punto di partenza per la navigazione del grafo. La memoria è implementata come una lista doppia, molto semplice da sviluppare e comprendere per verificarne la correttezza. Appartiene a questa classe la RedSIB volatile in RAM.
- "hash store" che fa uso di hash di chiavi associati al nodo centrale. Gli hash possono essere in memoria non persistente oppure persistente; mediante Berkeley DB (BDB) o Virtuoso (database management system che tramite licenza open source, possono essere utilizzati sulla maggior parte dei sistemi operativi). Appartengono a questa classe la RedSIBh, la RedSIB Berkeley DB e la RedSIB Virtuoso.

Virtuoso è principalmente un database SQL object-relational. Come database fornisce il meccanismo delle transazioni, un interprete SQL, stored-procedure, backup con un checkpoint sicuro ed inoltre supporta SPARQL per l'interrogazione di dati RDF memorizzati. Con virtuoso è possibile sviluppare soluzioni che utilizzano XML come livello fondamentale per l'accesso ai dati, tramite il quale viene fornito un accesso trasparente ai dati strutturati e non strutturati. In particolare i documenti XML possono essere generati sia interamente che dal web e memorizzarti nell'ecosistema di virtuoso. Virtuoso è progettato per

avvantaggiarsi del supporto multi-read e multi-processor, esso consiste in un singolo processo con un numero variabile di thread condivisi dai clients. Più thread possono cooperare sulla stessa struttura dati, causando la minima interferenza l'uno con l'altro. I thread condividono una cache delle pagine del database, mentre le vecchie pagine sono scritte su disco come processo in background a minima priorità. Tutte le tabelle del database sono conservate in un singolo insieme di file. Un insieme separato è invece usato per i file temporanei [35].

Berkeley DB offre le stesse funzionalità di archiviazione dei sistemi di database relazionali, come transazioni e recupero, multi-processor e multi-threading, backup e replica per le applicazioni ad alta concorrenza. Berkeley DB è in grado di gestire i database in memoria, su disco o in forma combinata ed è progettato per funzionare in modo completamente automatico, per garantire una gestione dell'amministrazione runtime, programmaticamente controllata. Berkeley DB offre una veloce, scalabile, affidabile biblioteca in un piccolo spazio [36].

4.4.2 CuteSIB

Questa SIB si basa su RedSIB e usa la sua medesima logica, è infatti scritto in C.Si tratta di un progetto recente, realizzato in Russia dal Laboratorio dell'università Petrozavodsk, la cui prima versione è stata resa disponibile il 17/08/2015. La principale novità introdotta di questa SIB è la rimozione di D-BUS, supporta plug-in, ovvero estensioni delle funzionalità originarie, relative alla parte di rete e fa uso delle libreria QT, una libreria multipiattaforma per lo sviluppo di programmi con interfaccia grafica [37].

4.4.3 SIB OSGi

OSGi è un'organizzazione no-profit, indipendente, che lavora per definire e promuovere specifiche per la fornitura di servizi per ambienti di rete. L'OSGi è considerato oggi come un "*Universal Middleware*"," *Software che si scrive una volta sola e quindi è utilizzabile in forma binaria universalmente: in molte diverse piattaforme, molti settori e per molti scopi diversi*" (Peter Kriens - OSGi Nucleo Platform Group). Si tratta di un sistema di servizio per piattaforme. Supporta il linguaggio di programmazione Java il quale implementa un completo e dinamico modello a componenti [38][39]. Il framework di OSGi:

- È semplice;
- È dinamico;
- È adattivo;
- Fornisce un'API completa per la gestione del ciclo di vita;

Tecnicamente le specifiche della piattaforma OSGi introducono il concetto bundle, inteso come archivio (JAR), contenente l'implementazione dei servizi e le direttive di distribuzione ed installazione all'interno della piattaforma, oltre che le dipendenze da altri package e servizi. Il framework OSGi è di fatto l'ambiente di esecuzione dei bundle. I bundle possono essere installati, avviati, fermati, aggiornati e disinstallati senza far cadere l'intero sistema. In questa caratteristica risiede la forte dinamicità [40]. Le soluzioni basate su OSGi sono facili da implementare, perché lo standard specifica come sono installati e gestiti i componenti. In particolare la SIB OSGi, realizzata dall'Università di Bologna in collaborazione con Eurotech (2011), sfrutta il framework appena descritto ed è costituita da diversi bundle. La sua architettura è basata sull'approccio OSGi e i suoi componenti sono configurati a partire dalle API. I Bundle che compongono la SIB OSGi sono:

- TCP Bundle;
- Message Handler Bundle;
- SSAP Bundle;
- Token Handler Bundle;
- Tools Bundle;
- SIB Bundle;
- Subscription Bundle;
- JOIN Bundle;
- Persistent Update Bundle.

Il modulo responsabile della gestione dell'interfacciamento con il mondo esterno è il TCP Bundle, il quale legge il messaggio ricevuto dai KP e vi aggiunge un riferimento per renderlo processabile dal Message Handler Bundle. Quest'ultimo delega il messaggio al SSAP Bundle il quale ha il compito di crearne una rappresentazione interna (a cui il Token Handler Bundle assegna un

identificativo univoco), che il SIB Bundle sfrutterà per processare la richiesta e fornire una risposta appropriata. SIB Bundle si appoggia alle librerie JENA, un framework Java di ultima generazione, per lo sviluppo di applicazioni orientate al web semantico. Il linguaggio JAVA e il framework OSGi rendono la SIB portabile su diversi sistemi operativi come Windows e MAC OS oltre a GNU/Linux.

4.5 Il benchmark LUBM

Vari sistemi della conoscenza che sono stati sviluppati per la memorizzazione, ragionamento e interrogazione del Web Semantico differiscono sulla base di diversi fattori come la memoria (volatile o persistente) o il grado di ragionamento. Generalmente ci sono due requisisti fondamentali per tali sistemi, la scalabilità e l'efficienza, associati a una capacità di ragionamento sufficientemente elevata da poterlo sostenere. Per valutarli sono necessarie estese dimensioni di dati e ricche ontologie. Il LUBM [41] estende query su un ampio set di dati, impiegato su un'ontologia dalle medie dimensioni e media complessità. Questa ontologia si basa su un dominio universitario e i suoi dati sono generati e poi installati su un dispositivo da valutare.

Il test nell'articolo [41] è realizzato per mettere a confronto differenti classi di sistemi, per i quali sono restituiti risultati sperimentali, ragionamenti ed osservazioni.

Il benchmark è stato creato con i seguenti obbiettivi:

1. Sostenere query estese, ovvero non solo query intenzionali (su classi e proprietà), ma queste ultime associate ad un uso dei dati intelligente.
2. Ottenere una scala arbitraria di dati, per rendere possibile la variazione delle dimensione dei dati
3. Basarsi su una ontologia di medie dimensioni e media complessità.

L'ontologia usata nel Benchmark è *Univ-Bench* la quale descrive università, uffici e le attività che si verificano al suo interno. L'ontologia include 43 classi e 32 proprietà (25 ObjectProperty e 7 DatatypeProperty).

4.5.1 Gerarchia delle classi

L'albero seguente mostra la gerarchia delle classi dell'ontologia del benchmark LUBM.

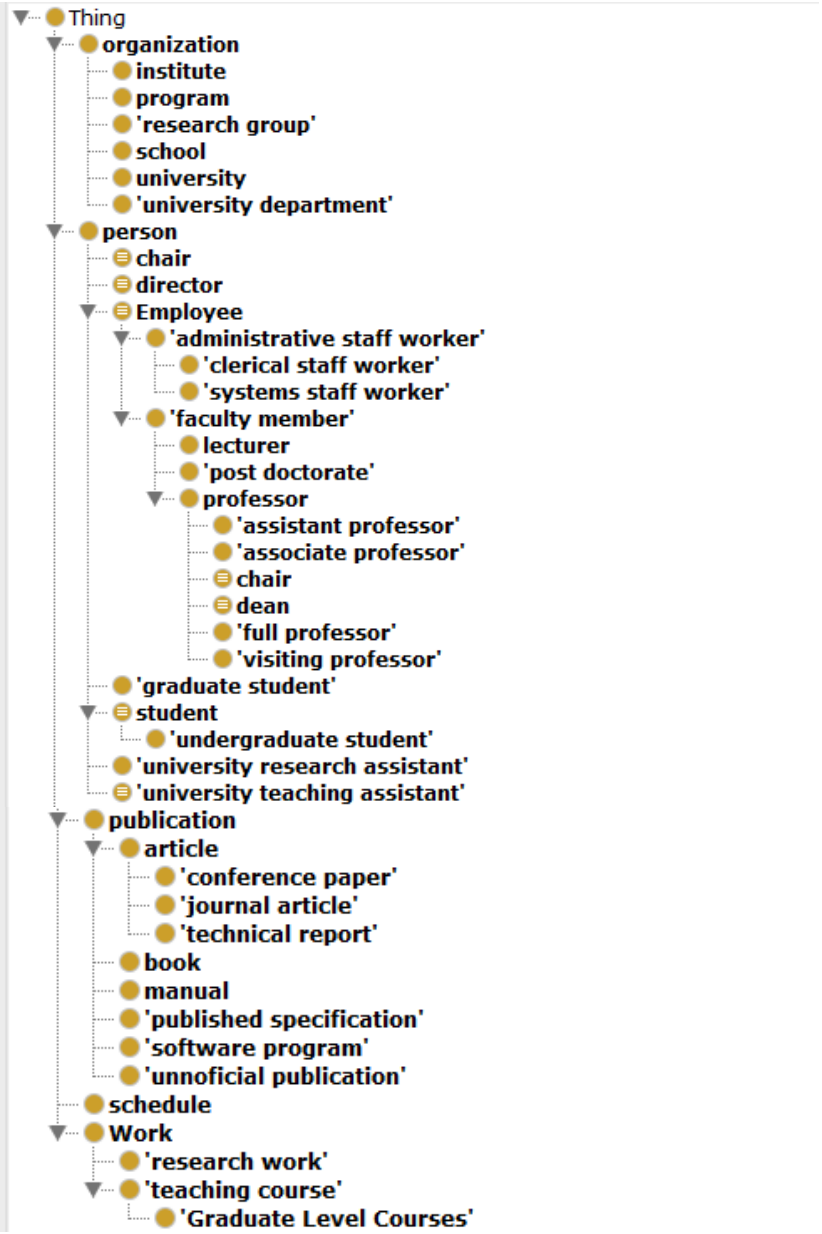


Figura 16 Gerarchia delle classi del benchmark LUBM

4.5.2 Datatype e object properties

Le tabelle seguenti mostrano le proprietà dell'ontologia suddivise per object properties e datatype properties.

Nome Object Property	Dominio	Ranges	Object Property padre
Has a degree from	person	university	
Has a doctoral degree from	person	university	Has a degree from
Has a masters degree from	Person	university	Has a degree from
Has an unfdergraduate degree from	person	university	Has a degree from
Has as a member	organization	person	
Has as a research project	research group	Research work	
Has as an alumnus	university	person	
Is a teaching assistant for	University teaching assistant	Teaching course	
Is about	publication	Research work	
Is affiliated with	organization	person	
Is affiliated with	organization	organization	
Is being advised by	person	professor	
Is documented in	software program	publication	
Is part of	organization	organization	
Is taking			
Is tenured	professor		
Is version	software program		
Lists as a course	schedule	Teaching course	
Member of	person	organization	
Works for	person	organization	Member of
publishes	organization	publication	
teaches	Faculty member	Teaching course	
Was written by	publication	person	
Was written on	publication		

Figura 17 Object properties del benchmark LUBM

Nome TopDataProperty	Dominio	TopDataProperty padre
Can be reached at	person	
Is researching		
name		
Is age	person	name
Office room No		
Telephone number	person	
title	person	

Figura 18 Datatype properties del benchmark LUBM

4.5.3 Set di query

Le 14 query (Q1..Q14) sono scritte in SPARQL per tenere conto dei seguenti fattori:

- Formato input (ovvero ampiezza dello spazio di ricerca)
- Selettività
- Complessità
- Informazioni sulla gerarchia
- Assunzione di deduzioni logiche

Sono state scelte una serie di query, che coprono una gamma di queste proprietà. Alcune hanno maggiore input, altre maggiore selettività, altre ancora sono in grado di dare informazioni su profondità e larghezza di una gerarchia. La tabella seguente mostra un'analisi delle varie query rispetto ai fattori sopra riportati:

	Ampio spazio di ricerca	**Alta selettività**	**Alta complessità**	**Gerarchia delle classi complessa**	**Reasoning**
Q1	x	x			
Q2	x	x	x		
Q3	x	x	x	x	
Q4		x		x	x
Q5		x		x	x
Q6		x			x
Q7		x		x	x
Q8		x	x	x	x
Q9		x		x	x
Q10					x
Q11					x
Q12			x		x
Q13		x	x		x
Q14		x			

Il codice SPARQL delle query è riportato in Appendice.

4.5.4 Metriche di performance

Il benchmark definisce inoltre le seguenti **metriche di performance** a cui faremo riferimento nella valutazione delle prestazioni della SIB:

- Tempo di caricamento

  È il tempo di memorizzazione dei dati, compreso il tempo trascorso in qualsiasi elaborazione analisi o ragionamento.
- Formato repository

  Dimensione risultante del repository dopo aver caricato i dati di riferimento. Questo dato è misurato solo per sistemi con memoria persistente.
- Tempo di risposta della query
- Completezza della query e solidità

  Un sistema è completo se genera tutte le risposte che vengono richieste, tuttavia nel web semantico risposte parziali saranno ugualmente accettabili, per questo si parla di grado di completezza. La solidità, invece, misura il grado di validità di ogni risposta.
- Metrico combinato

  È la combinazione fra tempo di risposta, completezza e solidità. Tale metrica può essere usata per fornire una classifica assoluta dei sistemi. È facilmente variabile e dovrebbe essere usata con una attenta considerazione dei parametri utilizzati nel calcolo.

5 Valutazione delle prestazioni

Per lo svolgimento dei test abbiamo creato un nostro personale ambiente di lavoro, basato sul modello espresso dal LUBM, ma sul quale sono state apportate diverse modifiche. Dopo un'attenta analisi dei file OWL di LUBM è emersa la necessità di aggiungere in tutti gli OWL (eccetto Univ-bench.owl) il seguente Namespace:

xml:base="http://swat.cse.lehigh.edu/onto/univ-bench.owl".

Inoltre, l'analisi degli OWL ci ha consentito di verificare che in LUBM si utilizzano molto i bNode, tipologia di dato non gestito dalla RedSIB, indicatore di un sicuro fallimento nell'inserimento di molte informazioni in tale SIB. I test sono stati effettuati sulla seguente configurazione:

- **Server** (sul quale vengono eseguite le varie SIB): 12 CPUs Intel Xenox CPU ES-2430 v2 @ 2.5 GHz 6 Cores, 64 GB di RAM, Ubuntu Server 15.04. Le SIB vengono eseguite all'interno di una macchina virtuale dedicata (Ubuntu 12.04 LTS, 32 GB RAM, 8 CPUs).
- **Client** (sul quale vengono lanciati i programmi di test sotto descritti): macchina virtuale VirtualBox, con sistema operativo Debian, con 2 GB di RAM e un processore (sistema host: portatile ASUS con processore Intel i7-3610QM (6 Cores) a 2.3 GHz, con 4 GB di RAM e sistema operativo Windows 7 Home Premium). Il client è collegato al server attraverso una connessione LAN a 100 Mbps.

Attraverso il terminale viene richiamato il programma "filler.py" e caricato il dataset di interesse nelle rispettive porte. Questo processo permette di ottenere le metriche relative al numero di triple che devono essere inserite (triple inserted), il numero effettivo di triple inserite (Triple into the SIB) e il tempo di caricamento (Elapsed time).

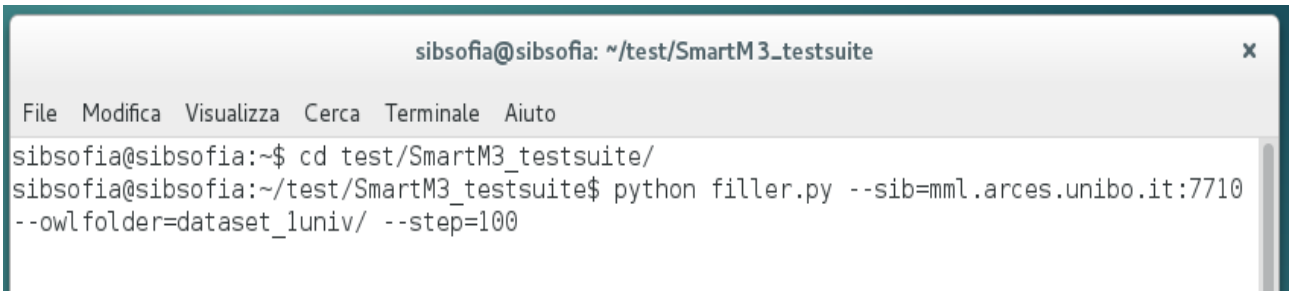


Figura 19 Richiamo del programma "filler.py" e caricamento dataset

In questo esempio viene caricata la SIB mml.arces.unibo.it:7710 con il dataset da una università (il parametro step consente di specificare il numero di triple da inserire ad ogni iterazione). Successivamente per l'esecuzione dei test viene richiamato il programma "query_test.py"

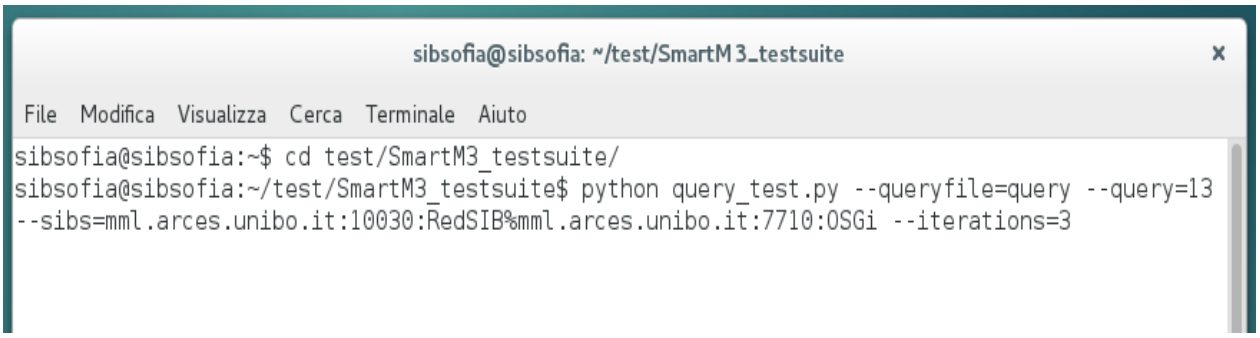


Figura 20 Richiamo del programma "query_test.py"

In questa dimostrazione viene caricato il file di query.py da cui verrà eseguita la query "Q13" per 3 volte. Le SIB da testare sono la mml.arces.unibo.it:10030 e la mml.arces.unibo.it:7701 a cui sono stati assegnati i nomi "RedSIB" e "OSGi".

È possibile anche usufruire del comando --help per avere una descrizione completa del suo utilizzo.

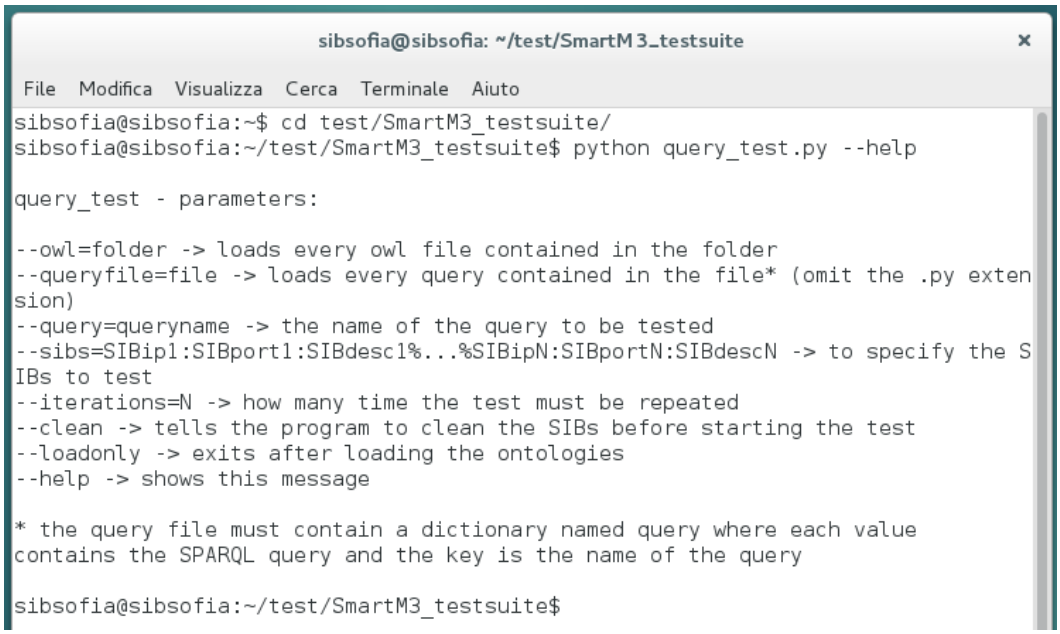


Figura 21 rappresentazione comando --help

I test sono stati eseguiti nel seguente ordine:

1. Sono state valutate le quattro SIB non persistenti: SIB OSGi, RedSIB, RedSIBh e CuteSIB prima con il dataset LUBM ad una università, poi su cinque.
2. E tre SIB persistenti: SIB OSGi, RedSIBBDB e RedSIBVirtuoso. Con il medesimo ordine di dataset.

5.1 Sintesi dei risultati

Nel seguente paragrafo è riportata la sintesi dei risultati ottenuti dall'esecuzione dei test precedentemente descritti.

	LUBM (1,0)		LUBM (5,0)	
SIB	**Triple inserite***	**Tempo**	**Triple inserite****	**Tempo**
OSGi	100837	≈ 1 minuto	624826	≈ 5 minuti
RedSIB	101398	≈ 6 minuti	624948	≈ 5 ore
RedSIBh	100968	≈ 1 minuto	624949	≈ 5 minuti
CuteSIB	100980	≈ 2 minuti	624945	≈ 13 minuti
* Il numero totale di triple da inserire è pari a 103898				
** Il numero totale di triple da inserire è pari a 647855				

Figura 22 Tempi di caricamento con dataset da 1 e 5 università (LUBM(1,0) e LUBM(5,0)) per il caso non persistente

	LUBM (1,0)		LUBM (5,0)	
SIB	**Triple inserite***	**Tempo**	**Triple inserite****	**Tempo**
OSGi	100835	≈ 6 minuti	624826	≈ 45 minuti
RedSIBDBD	100969	≈ 30 secondi	624945	≈ 4 minuti
RedSIBVirtuoso	100965	≈ 3 minuti	624944	≈ 20 minuti
* Il numero totale di triple da inserire è pari a 103898				
** Il numero totale di triple da inserire è pari a 647855				

Figura 23 Tempi di caricamento con dataset da 1 e 5 università (LUBM(1,0) e LUBM(5,0)) per il caso persistente

	OSGi		RedSIB		RedSIBh		CuteSIB	
Query	Tempo (s)	N° risultati	Tempo (s)	N° risultati	Tempo (s)	N° risultati	Tempo (s)	N° risultati
Q1	0.150	4	585	4	x	x	x	x
Q2	0.030	0	x	x	x	x	x	x
Q3	0.020	6	x	x	x	x	x	x
Q4	0.013	0	0.045	0	0.012	0	0.014	0
Q5	0.008	0	0.039	0	0.013	0	0.008	0
Q6	0.012	0	0.051	0	0.013	0	0.009	0
Q7	0.009	0	0.040	0	0.011	0	0.008	0
Q8	0.012	0	0.054	0	0.014	0	0.007	0
Q9	0.009	0	0.037	0	0.013	0	0.008	0
Q10	0.013	0	0.051	0	0.013	0	0.013	0
Q11	0.012	0	7.044	0	49.422	0	x	x
Q12	0.014	0	0.047	0	0.008	0	0.006	0
Q13	0.012	0	0.039	0	0.008	0	0.008	0
Q14	0.890	5916	0.787	5916	0.75	5916	4.299	5916
x = la query ha provocato un errore irreversibile ; T = la query non ha risposto entro 5 minuti								

Figura 25 Tempi di risposta delle queries per il dataset da 1 università, non persistente.

	OSGi		RedSIB		RedSIBh		CuteSIB	
Query	Tempo (s)	N° risultati	Tempo (s)	N° risultati	Tempo (s)	N° risultati	Tempo (s)	N° risultati
Q1	0.189	4	x	x	x	x	x	x
Q2	0.104	0	x	x	T	x	T	x
Q3	0.076	6	x	x	x	x	x	x
Q4	0.015	0	0.163	0	0.011	0	0.011	0
Q5	0.013	0	0.151	0	0.006	0	0.009	0
Q6	0.012	0	0.150	0	0.010	0	0.012	0
Q7	0.014	0	0.150	0	0.009	0	0.012	0
Q8	0.014	0	0.152	0	0.009	0	0.011	0
Q9	0.014	0	0.152	0	0.010	0	0.012	0
Q10	0.013	0	0.153	0	0.008	0	0.010	0
Q11	0.013	0	x	x	x	x	T	x
Q12	0.014	0	0.161	0	0.008	0	0.011	0
Q13	0.013	0	0.160	0	0.010	0	0.013	0
Q14	4.263	36682	5.015	36682	5.325	36682	28.083	36682
x = la query ha provocato un errore irreversibile ; T = la query non ha risposto entro 5 minuti								

Figura 24 Tempi di risposta delle queries per il dataset da 5 università, non persistente

	OSGi		RedSIBBDB		RedSIBVirtuoso	
Query	Tempo (s)	N° risultati	Tempo (s)	N° risultati	Tempo (s)	N° risultati
Q1	0.021	4	T	X	1.215	4
Q2	0.079	0	X	X	X	X
Q3	0.068	6	X	X	4.298	6
Q4	0.024	0	0.025	0	0.018	0
Q5	0.017	0	0.01	0	0.012	0
Q6	0.016	0	0.011	0	0.012	0
Q7	0.018	0	0.009	0	0.012	0
Q8	0.016	0	0.008	0	0.009	0
Q9	0.019	0	0.010	0	0.012	0
Q10	0.016	0	0.009	0	0.011	0
Q11	0.028	0	58.026	0	0.177	0
Q12	0.016	0	0.010	0	0.012	0
Q13	0.017	0	0.010	0	0.012	0
Q14	0.813	5916	0.827	5916	0.900	5916
x = la query ha provocato un errore irreversibile ; T = la query non ha risposto entro 5 minuti						

Figura 26 Tempi di risposta delle queries per il dataset da 1 università, persistente.

	OSGi		RedSIBBDB		RedSIBVirtuoso	
Query	Tempo (s)	N° risultati	Tempo (s)	N° risultati	Tempo (s)	N° risultati
Q1	0.095	4	X	X	8.523	4
Q2	0.187	9	X	X	X	X
Q3	0.100	0	X	X	26.461	6
Q4	0.016	0	0.009	0	0.016	0
Q5	0.014	0	0.009	0	0.012	0
Q6	0.012	0	0.009	0	0.010	0
Q7	0.016	0	0.010	0	0.012	0
Q8	0.014	0	0.009	0	0.010	0
Q9	0.014	0	0.010	0	0.012	0
Q10	0.014	0	0.008	0	0.009	0
Q11	0.023	0	X	0	0.997	0
Q12	0.015	0	0.009	0	0.012	0
Q13	0.014	0	0.01	0	0.011	0
Q14	4.111	36682	4.688	36682	6.127	36682
x = la query ha provocato un errore irreversibile ; T = la query non ha risposto entro 5 minuti						

Figura 27 Tempi di risposta delle queries per il dataset da 5 università, persistente.

5.2 Analisi risultati

5.2.1 SIB non persistenti

5.2.1.1 Dataset da 1 univeristà

1. **Tempo di caricamento**: i test di caricamento del dataset sulla piattaforma hanno riportato in ordine di tempistica decrescente i seguenti risultati: OSGi è stata la più rapida seguita da RedSIBh, CuteSIB e RedSIB. Le prime tre hanno caricato un numero di triple circa equivalente, intorno alle 100800 triple, seguite da RedSIB che ne ha riportato un numero maggiore di circa 101400 triple. Il suo tempo elevato di caricamento dunque non compensa e giustifica la completezza del sistema.

2. **Tempo di riposta delle query**: proseguendo in ordine, la Q1 è stata eseguita correttamente solo da OSGi e RedSIB, tra le quali il numero di risposte risulta coerente. Tra le due i tempi sono molto diversi, 0.15 s per OSGi e 585 s per RedSIB. Le restanti SIB non hanno riportato risultati incorrendo in errori. Una breve considerazione va fatta in merito al RedSIB, che come forma base sulla quale la cuteSiB e RedSIBh nascono, è stata l'unica fra questo complesso a riportare risultati seppur scadenti.

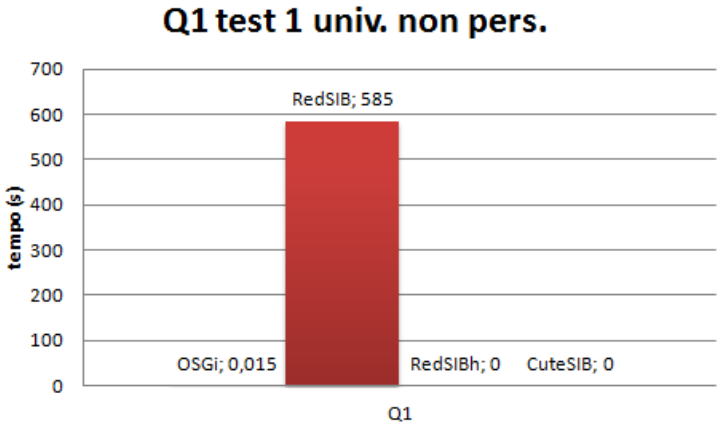


Figura 28 Q1 per LUBM(1,0) (caso non persistente)

Q2 e Q3 sono state completate con successo solo da OSGi in un tempo minimo.

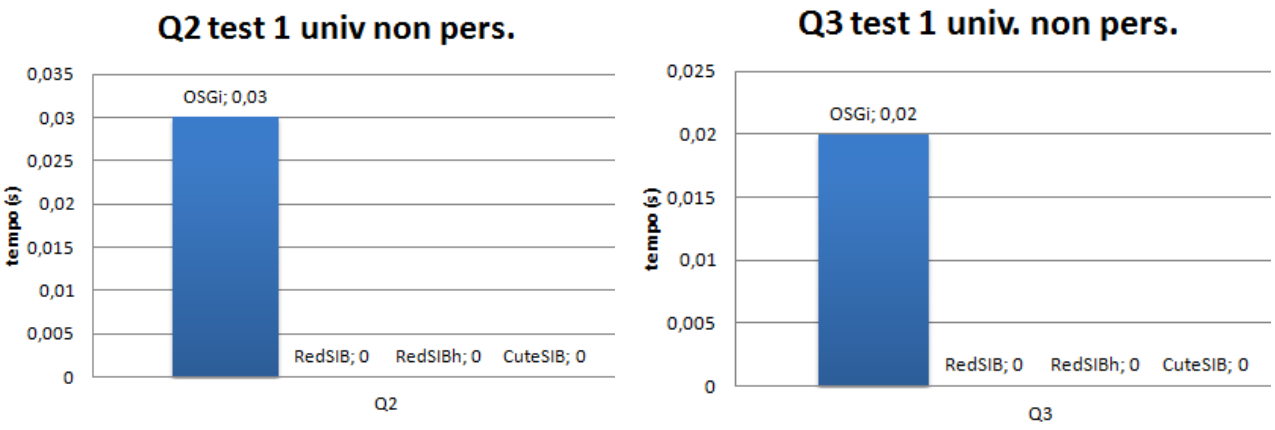
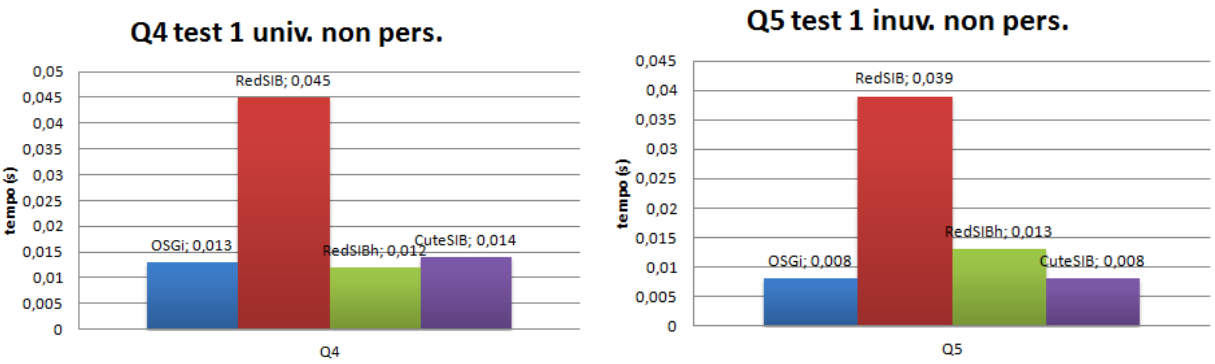


Figura 29 Query Q2, Q3 con LUBM(1,0) (caso non persisstente)

Questa generale limitazione per le query Q1, Q2 e Q3 potrebbe essere causata dalla grande quantità di dati in ingresso limitati a poche classi (tra cui "Graduate Student"), che tali query prevedono, accompagnate da richieste altamente selettive in uscita. Probabilmente l'interrogazione del dataset si dilata nel tempo per quelle piattaforme che usano la struttura di memorizzazione tipica di RedSIB, a tal punto da causare perdite di risultati.

I fallimenti di queste query tuttavia non si limitano solo all'assenza di dati materiali in uscita, ma si estendono in un contesto ben più ampio. L'errore comporta un blocco delle SIB interessate, le quali smettono di funzionare nella loro totalità. In un contesto reale ciò comporterebbe numerosi danni, con un peso tanto maggiore quanto rilevante è l'applicazione che ne fa uso.

Per quanto concerne le query dalla Q4 alla Q10, il lasso di tempo impiegato da OSGi, RedSIBh e CuteSIB è circa equivalente e si aggira intorno agli 0.01 s. Diverso è il discorso per RedSIB, che come si poteva giustamente prevedere ha mostrato prestazioni peggiori con tempistiche intorno agli 0.04-0.05 s.



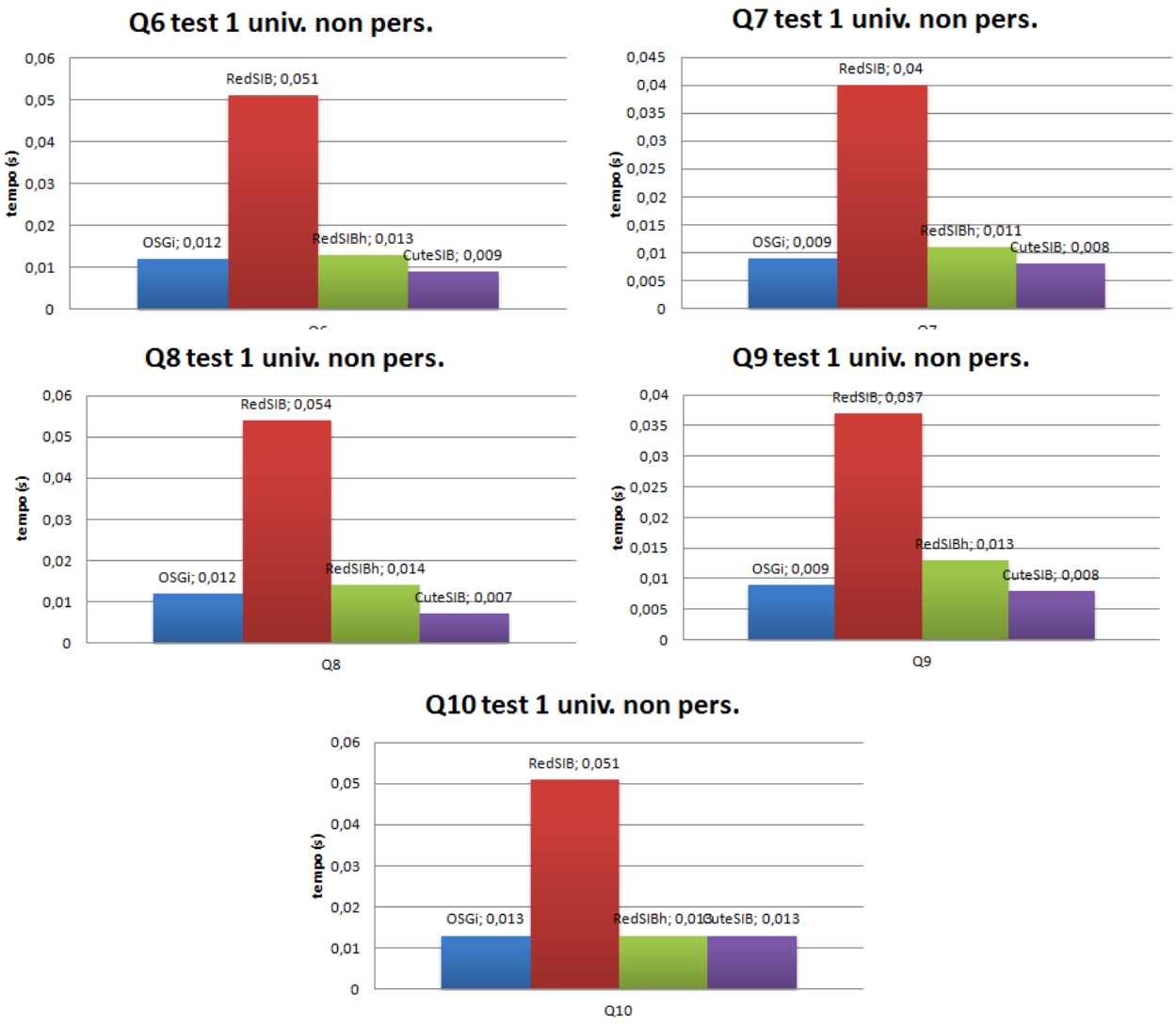


Figura 30 Da Q4 a Q10 con LUBM(1,0) (caso non persistente)

Risulta naturale quindi domandarsi quali differenze rilevanti mostra questo insieme di query. La Q4 e la Q5 sono incentrate su indagini gerarchiche. Si interrogano infatti su proprietà multiple di una singola o più classi. Prevedono un ingresso limitato, ma comunque alta selettività. Presumibilmente l'elevata selettività è problematica solo in ampi dataset.

Ancora, le query dalla Q6 alla Q9 incrementano il numero di dati in ingresso, con risultati sempre più selettivi. La Q9 in particolare è simile alla Q2 e si differenzia solo per le diverse classi interrogate. Ciò è a favore delle deduzioni fatte in precedenza sul fallimento delle prime query. Infine la Q10 è interamente incentrata su relazioni implicite, oggetto di interesse per test sul reasoning, da noi non effettuati, dunque nel nostro contesto poco complessa.

La Q11 risulta interessante, con tempi e comportamenti di esecuzione disparati per le diverse SIB. In particolare CuteSIB fallisce non riportando alcun risultato, mentre RedSIBh impiega circa 49 s contro i 7 s di RedSIB e 0.012 s della OSGi. Le query Q12 e Q13 avvengono con successo e con un comportamento analogo al caso delle query dalla Q4 alla Q10. Ed infine la Q14 avviene con successo e tempistiche circa equivalenti per tutte le SIB intorno a 0.8 s. Considerando che le query comprese dalla Q11 alla Q14 sono basate sul reasoning è prevedibile il comportamento verificato dalle Q12 e Q13. La Q14 è la più semplice del set di dati con un ampio ingresso, bassa selettività e nessuna informazione gerarchica e deduttiva.

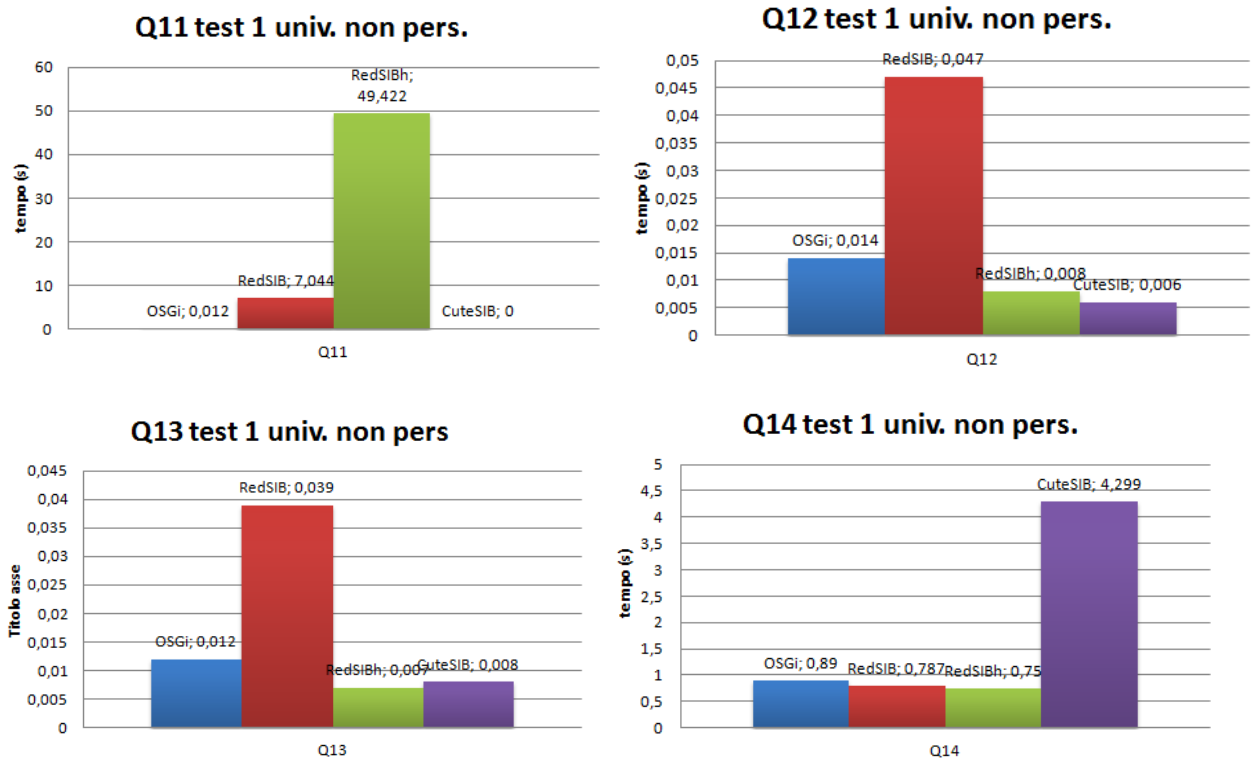


Figura 31 Da Q11 a Q14 con LUBM(1.0) (caso non persistente)

I tempi elevati sono dovuti alla tempo di risposta nel processo di comunicazione tra SIB e KP. La comunicazione fra questi due componenti può essere suddivisa in tre parti e quindi come somme di tre intervalli temporali. Il primo è il tempo di richiesta (t_req), ovvero il tempo di trasferimento della query dal KP alla SIB. Questo è circa equivalente per tutte le query, essendo quest'ultime di lunghezza conforme. Il secondo tempo è quello di elaborazione (t_elab) nel quale rientra tutto il complesso di lettura ed esecuzione della query. Infine il terzo ed ultimo è quello di risposta (t_ans) dedicato al trasferimento dei risultati dalla SIB al KP. Fatte queste considerazioni, la Q14 con 5916 risultati, rassegna tempi maggiori a

causa di un t_ans molto dispendioso. Per la Query 11 occorre un'analisi più approfondita, sebbene presenti delle classi già invocate nelle query precedenti e quindi non introduca nessuna novità. Probabilmente il pattern relativo a tale classe nelle query già viste era disposto in un ordine tale da non renderla un problema.

5.2.1.2 Dataset da 5 università

1. **Tempo di caricamento**: OSGi e RedSIBh impiegano circa 31000 ms a caricare l'intero dataset, contro i 18274031 ms adoperati da RedSIB. Questo distacco temporale, desume ancora una volta, come all'aumentare del carico OSGi si mostri sempre più performante a pari passo con RedSIBh. Nella parte intermedia di questa classifica temporale troviamo invece CuteSIB con 823971 ms.

2. **Tempo di risposta delle query**: le query Q1, Q2 e Q3 sono state eseguite con successo solo per OSGi e riportato ancora una volta errore per RedSIB, RedSIBh e CuteSIB. Con un'unica differenza in Q2 dove RedSIBh e CuteSIB sono andate in timeout.

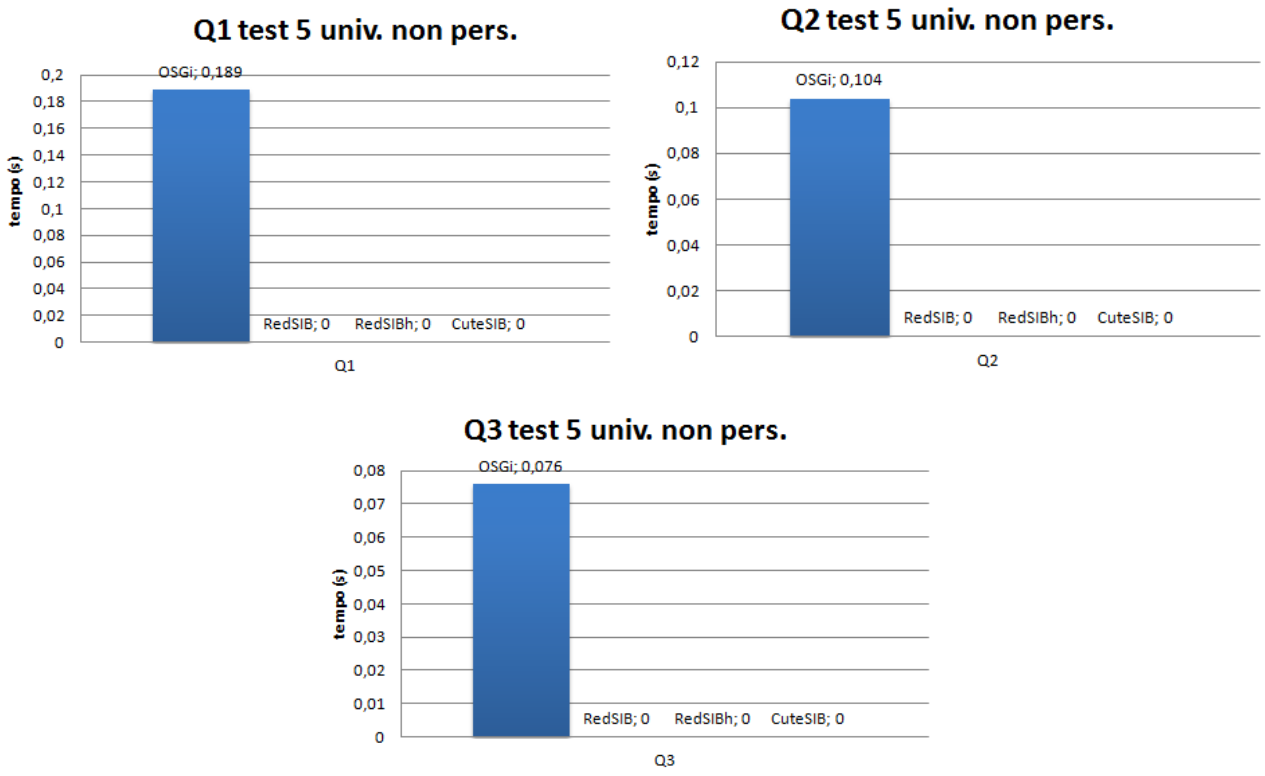
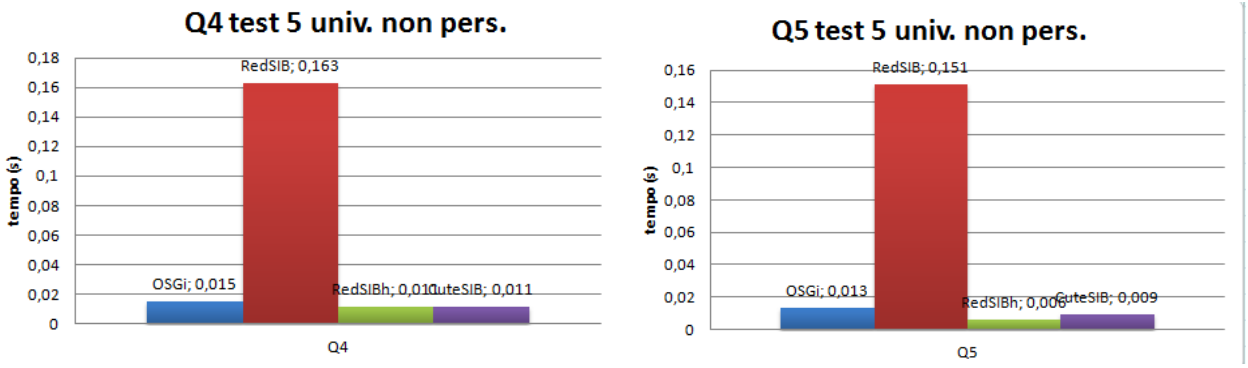
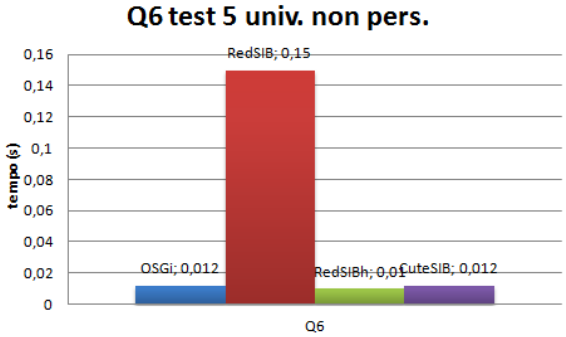


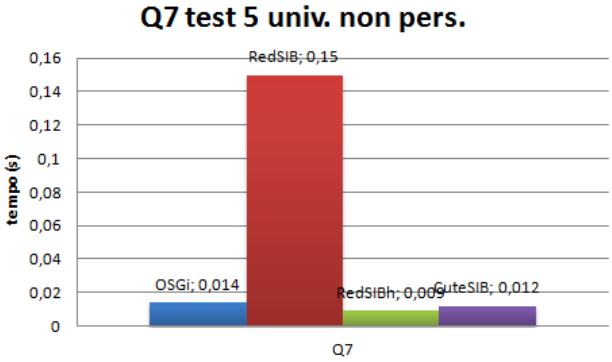
Figura 32 Da Q1 a Q3 LUBM(5,0) (caso non persistente)

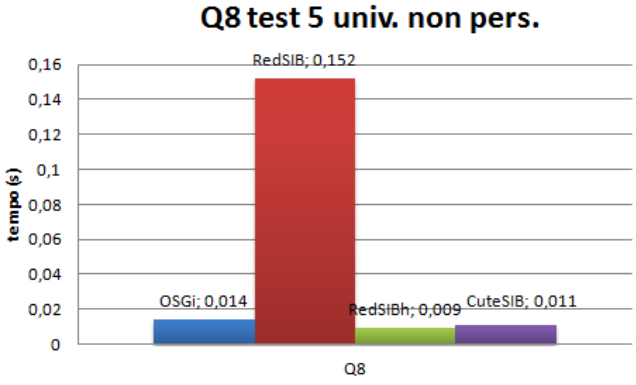
Per timeout si intende l'assenza di risposta entro un tempo prefissato, nel nostro caso pari a 5 min e oltre al quale le prestazioni non possono essere considerate accettabili. Ciò non significa che il sistema fosse destinato ad incorrere certamente in errore, ma in un reale caso d'uso ciò avrebbe manifestato una grave inefficienza.

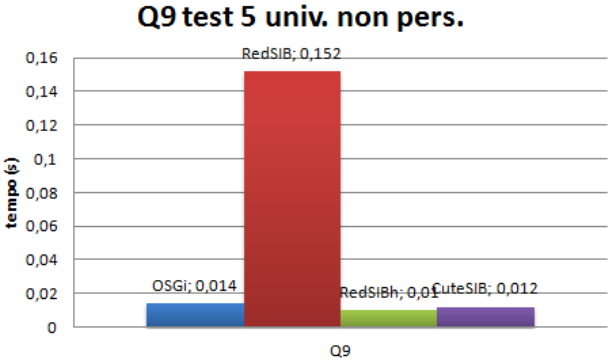
Le Query dalla Q4 alla Q10 ancora una volta mostrano ottimi risultati per OSGi, RedSIBh e CuteSIB, con valori attorno a 0.013 s, mentre RedSIB anche se funzionante si allunga con circa 0.15 s. L'aumento del dataset è stato molto più significativo per RedSIB che per OSGi o le altre SIB.











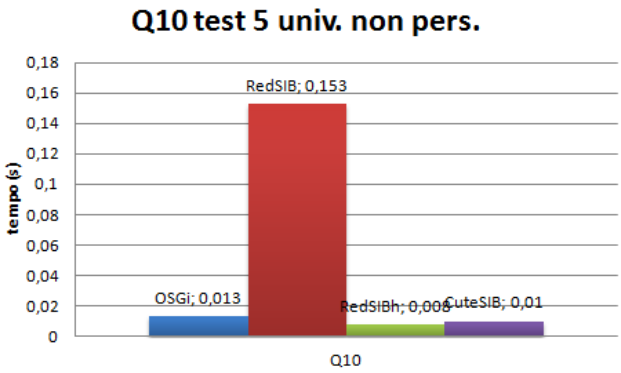


Figura 33 Da Q4 a Q10 con LUBM(5,0) (caso non persistente)

Alla Q11 ha risposto efficientemente solo OSGi, mentre RedSIB e RedSIBh hanno riportato errore. La risposta di CuteSIB è invece andata in timeout. Le Query Q13 e Q14 sono paragonabili alle Q4-Q10. Infine la Q14 è riuscita con successo per tutte le SIB, ma con tempi di circa 5 s per OSGi, RedSIB e RedSIBh, contro i 28 s adoperati da CuteSIB.

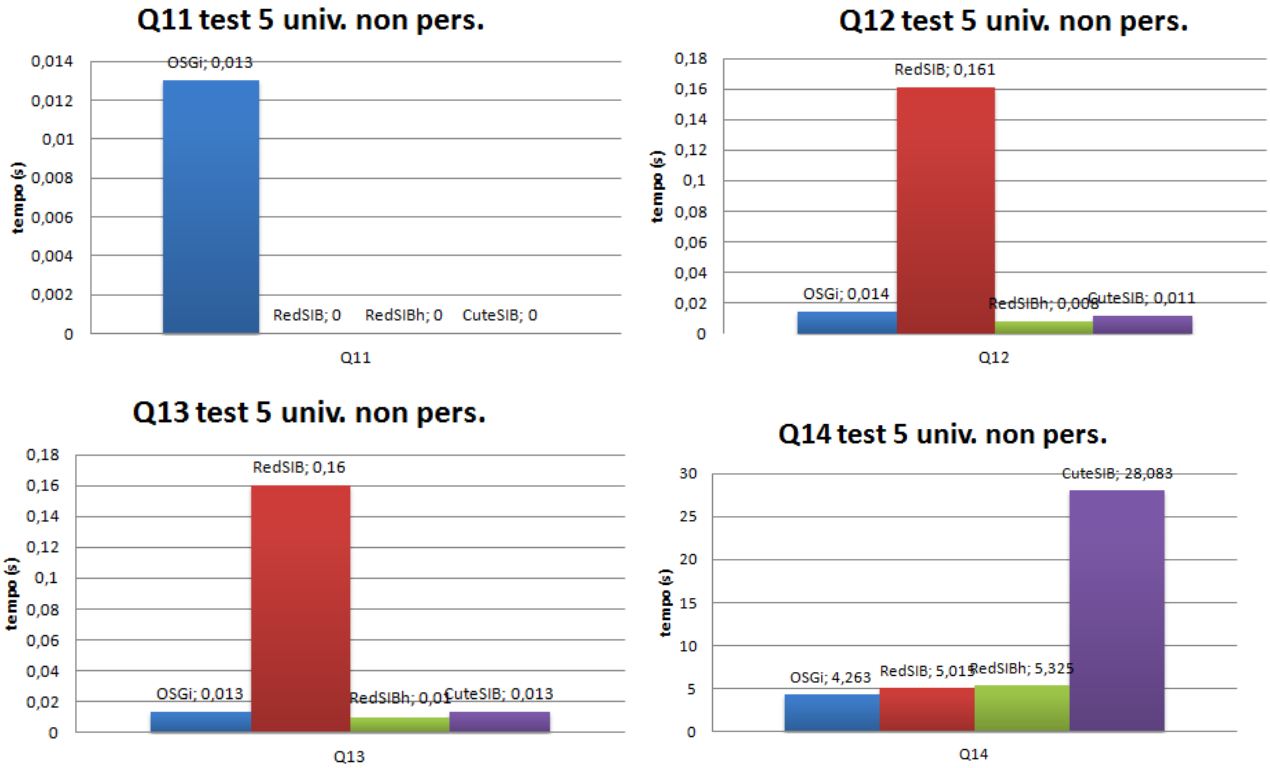


Figura 34 Da Q11 a Q14 con LUBM(5,0) (caso non persistete)

Le prestazioni di CuteSIB sono evidentemente rallentate dal numero di risultati, nello specifico 36682. Il tempo di risposta e trasferimento dei risultati è dunque rilevante.

5.2.1.3 Considerazioni finali

Complessivamente OSGi, sia in termini di caricamento, che di query, si evince essere la più performante ed in termini di solidità e completezza equivalente. Plausibilmente ciò è dovuto alla sua implementazione e il suo appoggio alle librerie JENA. Segue poi RedSIBh che nonostante diversi errori è comunque performante nel caricamento del data set e nelle query più semplici. Ed infine RedSIB e CuteSIB dalle pari prestazioni. CuteSIB più rapida, ma meno solida e completa nel numero di query eseguite, RedSIB più lenta, ma di maggiore completezza.

5.2.2 SIB persistenti

5.2.2.1 Dataset da 1 università

1. **Tempo di caricamento**: Nel contesto persistente i tempi di caricamento sono stati complessivamente maggiori. La più rapida è stata la RedSIBBDB (33052 s) seguita dalla RedSIBVirtuoso ed infine dalla OSGi con 372017 s.

   Già si può desumere un comportamento poco performante di OSGi in termini persistenti, ciò deriva dalla natura di OSGi nata come non persistente e quindi non ancora sviluppata efficientemente in questo contesto. Inoltre possiamo prevedere da parte di OSGi un numero di accessi al disco elevato, certamente causa di rallentamento.

2. **Tempo di risposta delle query**: la Query Q1 è stata eseguita con successo dalla OSGi e della RedSIBVirtuoso, ma ha dato timeout per la RedSIBBDB. I tempi ottenuti presentano la differenza di un secondo abbondante a favore della OSGi. La Q2 è avvenuta con successo solo per la OSGi e ha riportato errore per le restanti due SIB, mentre la Q3 ha un comportamento assimilabile alla Q1, ma con tempistiche diverse di 0.068 s per la OSGi e 4.3 s per la RedSIBVirtuoso.

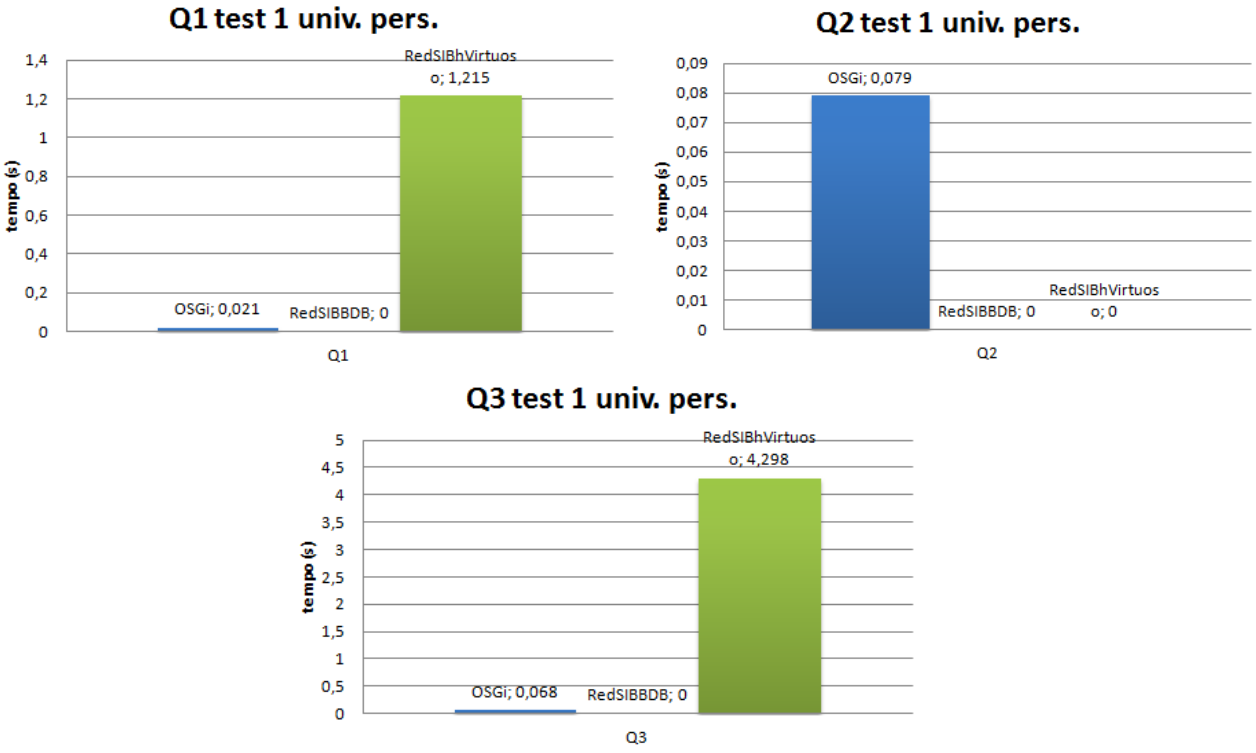


Figura 35 Dalla Q1 alla Q3 con LUBM (1,0) (caso persistente)

Le query dalla Q4 alla Q10 mostrano risultati similari. Tuttavia i tempi più bassi appartengono ancora alle RedSIBVirtuoso e RedSIBBDB.

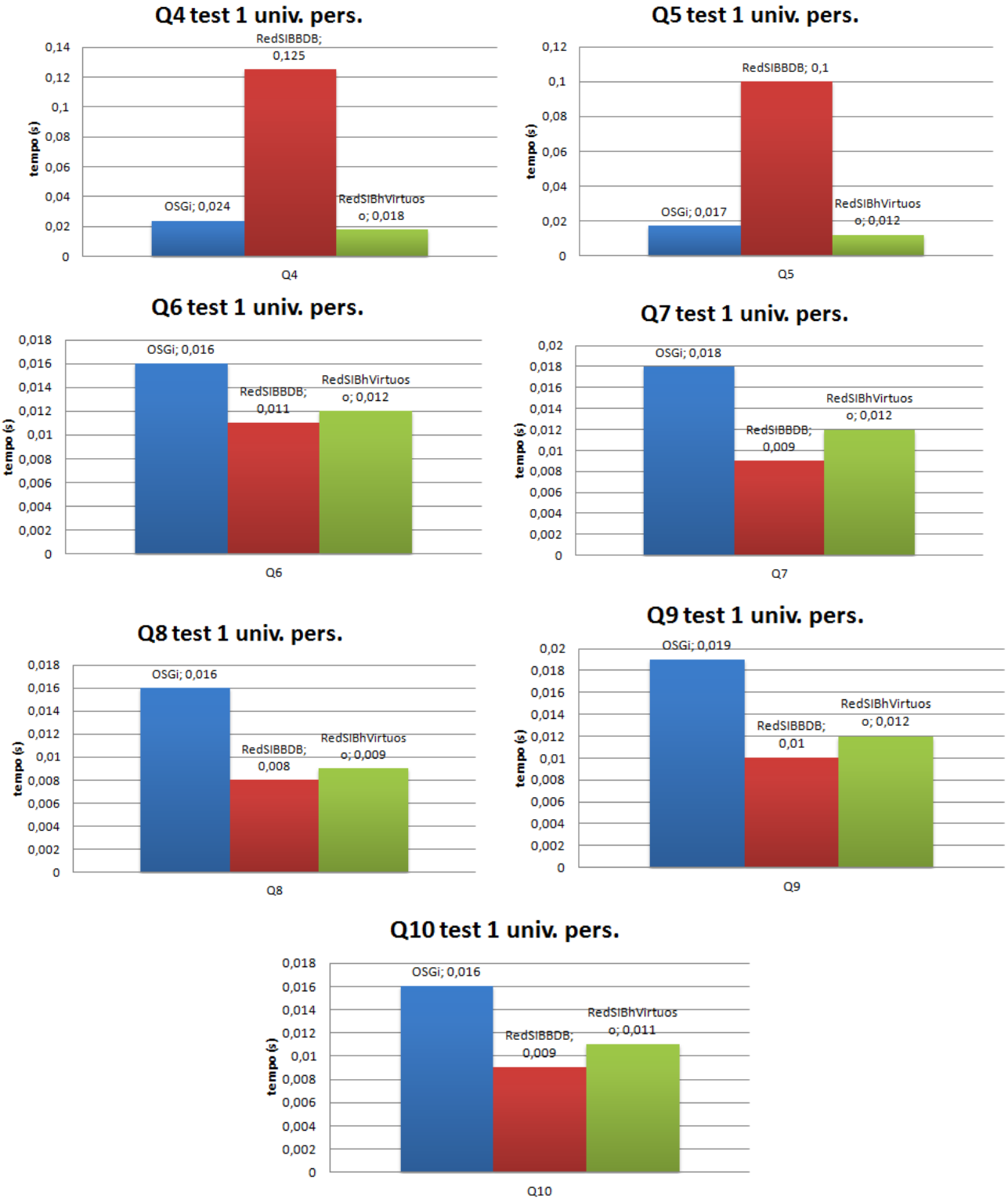


Figura 36 Dalla Q4 alla Q10 con LUBM (1,0) (caso persistente)

La Query Q11 è eseguita con un esito positivo per OSGi con 0.028 s, buono per RedSIBVirtuoso con 0.018 s e piuttosto svantaggioso per RedSIBBDB con 58 s. Q12 e Q13 ancora una volta sono equiparabili alle Q4-Q10 ed in conclusione la Q14 con circa 0.9 s è eseguita da tutte le SIB.

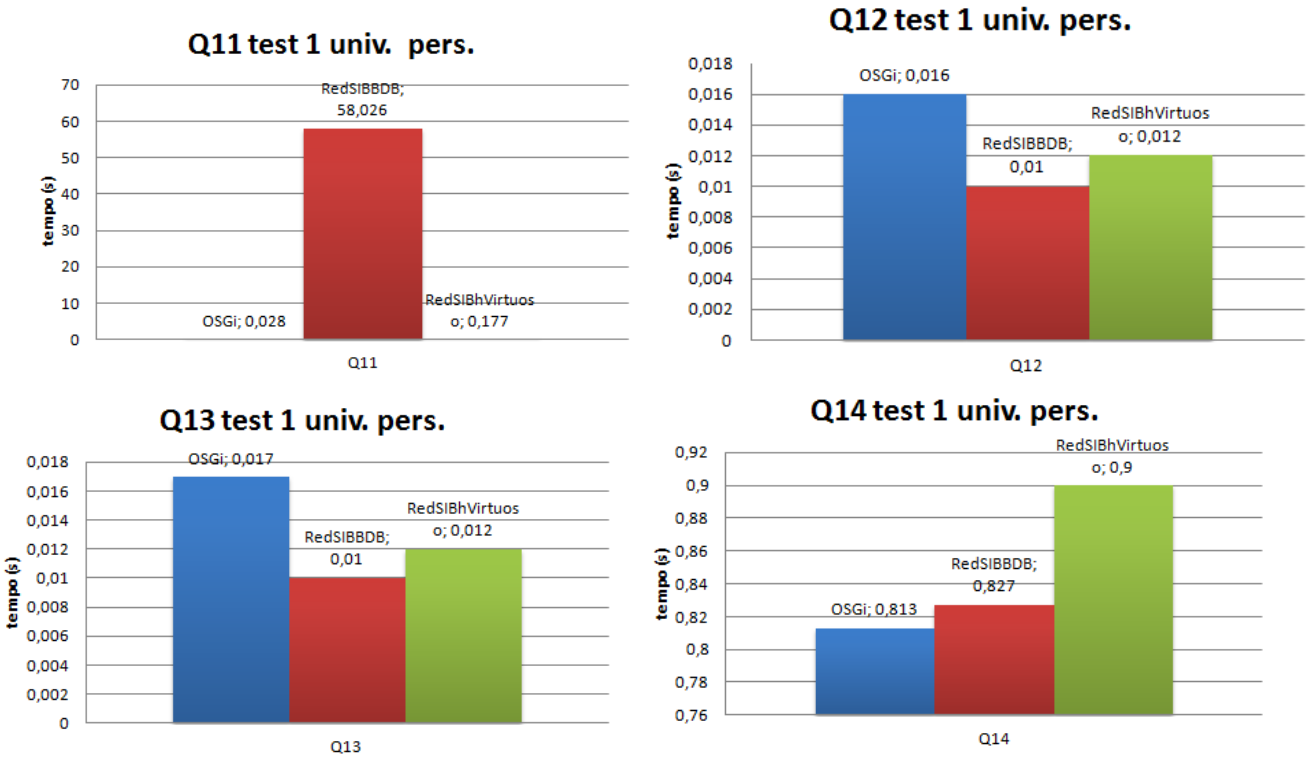


Figura 37 Dalla Q11 alla Q14 con LUBM (1,0) (caso persistente)

È evidente che in caso di difficoltà e quindi di complessità nella risoluzione delle query, OSGi ne esce sempre con maggiore successo e perfomance, ma di fronte a query elementari in termini di ingressi o comunque maggiormente incentrate su valutazioni gerarchiche, RedSIBVirtuoso e RedSIBBDB risultano essere migliori.

5.2.2.2 Dataset da 5 università

1. Tempo di caricamento: I Risultati in questione replicano in ordine quelli dei test ad 1 università. Ovviamente con tempi maggiori. Al primo posto troviamo quindi la RedSIBBDB seguita dalla RedSIBVirtuoso e OSGi. Una rilevante considerazione va fatta in merito al caricamento di RedSIBVirtuoso che non è stato eseguito interamene con successo. Infatti sono state caricate solo 647827 triple sulle 647855 totali.

Tempo di risposta delle query: la Query Q1 ha dato soluzione per le sole OSGi con 0.095 s e RedSIBVirtuoso con 8.52 s, mentre RedSIBBDB riporta errore. La Q2 è avvenuta con esito positivo solo per la OSGi e ha restituito errore per le restanti SIB.Per la Q3 invece il comportamento è analogo alla Q1 con 0.1 s per la OSGi e 26,4 s per la RedSIBBDB.

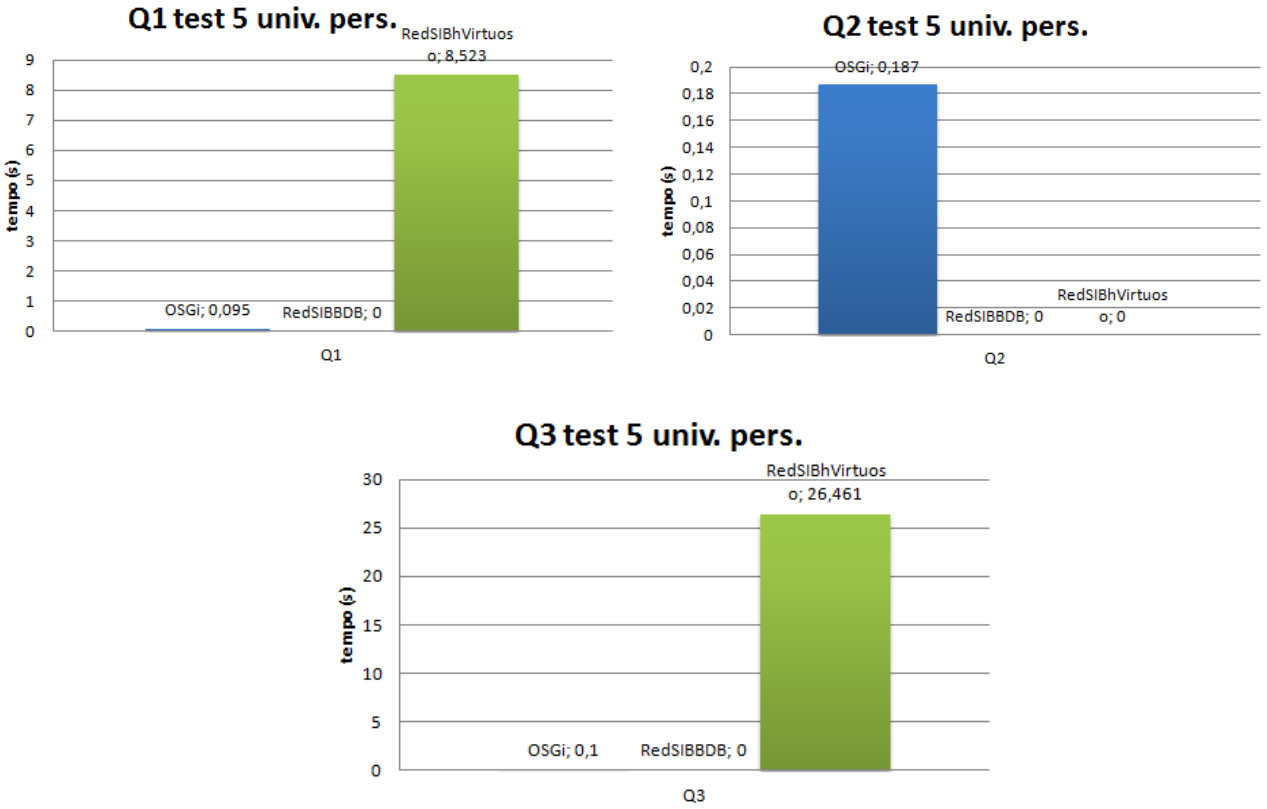


Figura 38 Dalla Q1 alla Q3 con LUBM (5,0) (caso persistente)

Dalla Query Q4 alla Q10 i risultati temporali sono uguagliabili con una lieve maggiore rapidità per la RedSIBBDB.

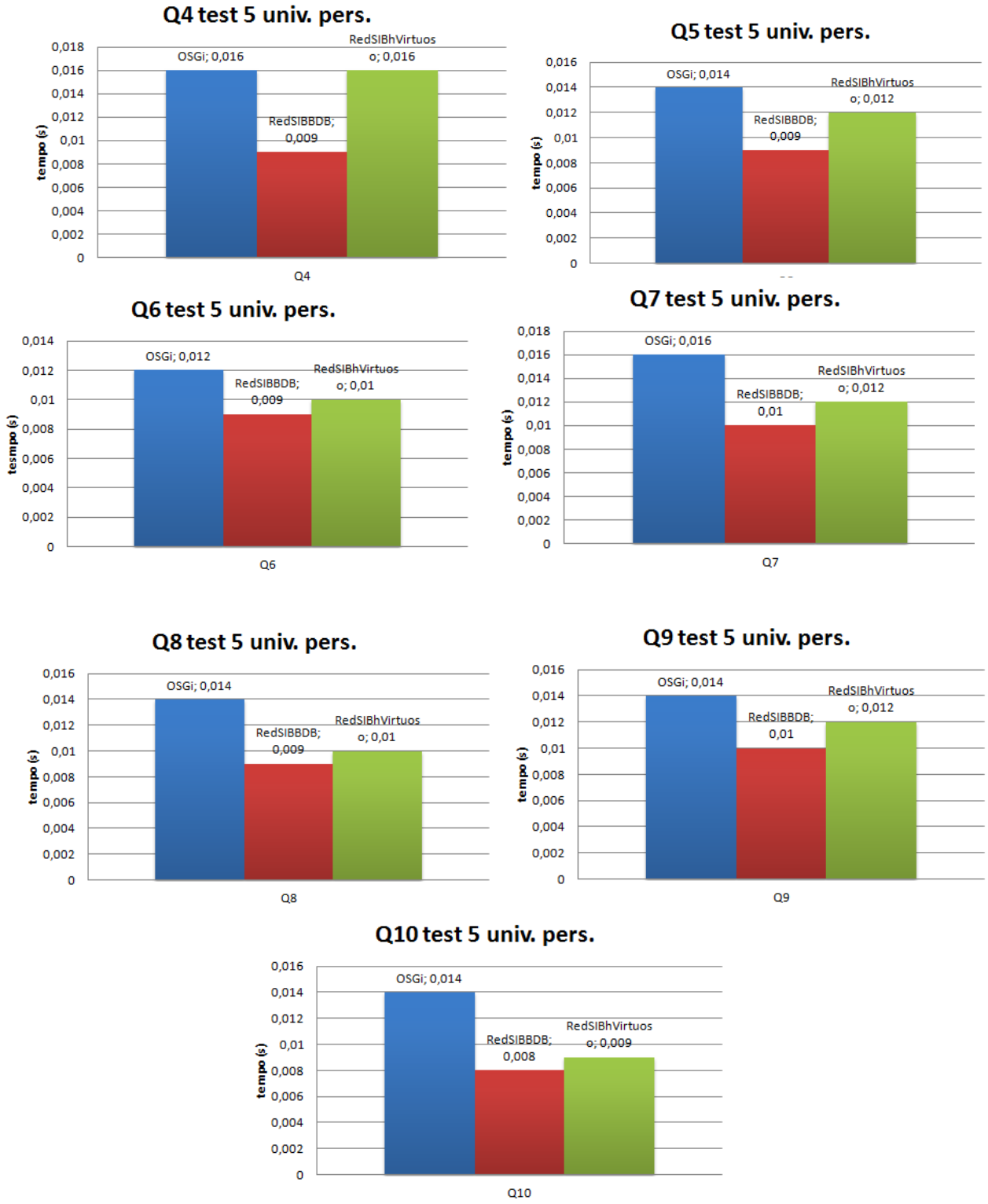


Figura 39 Dalla Q4 alla Q10 con LUBM (5,0) (caso persistente)

La Q11 fallisce solo per RedSIBBDB e viene eseguita regolarmente dalle restanti due SIB anche se tuttavia con tempistiche disparate. Infine le Query Q12 e Q13 si comportano come le query Q4-Q10 e la Q14 ha una totale riuscita con tempi di 4 s per OSGi e 6 s per RedSIBVirtuoso.

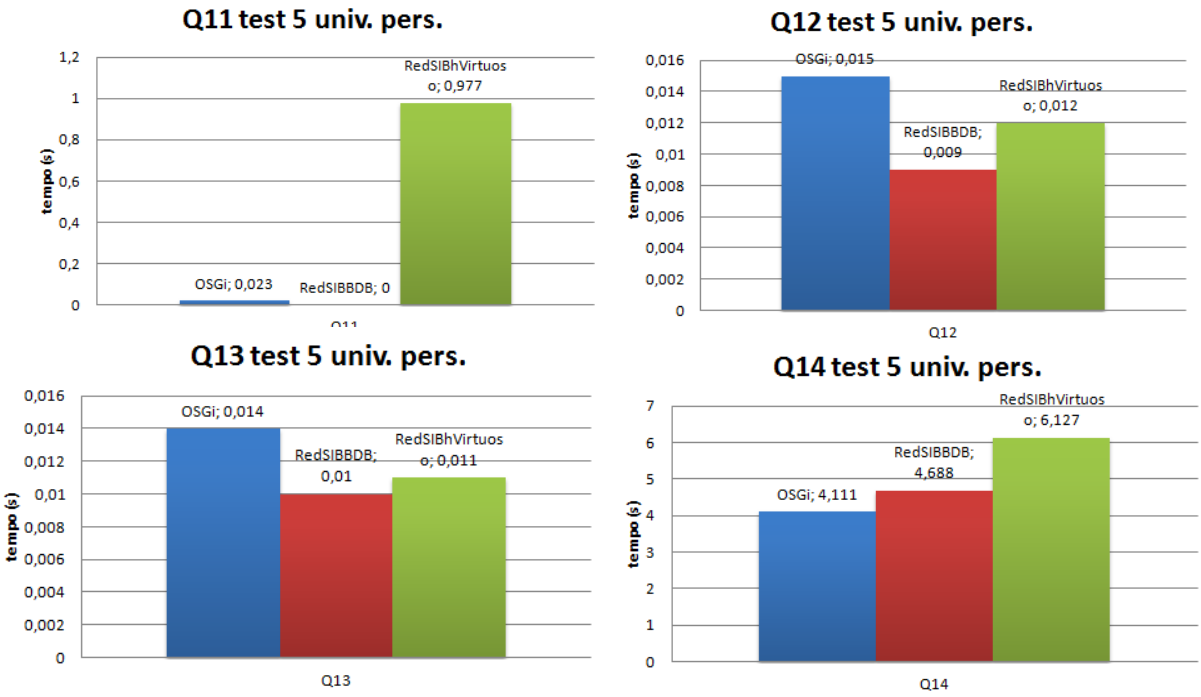


Figura 40 Dalla Q11 alla Q14 con LUBM (5,0) (caso persistente)

5.2.2.3 Dimensione del datastore

Per RedSIB Virtuoso è riportata una "X" in quanto non è possibile tutt'ora ricavare tale metrica [36].

Dimensione dello storage per il dataset da 1 universita:

- RedSIB BDB: 77 MB
- RedSIB Virtuoso: X
- OSGi: 14 MB

Dimensione dello storage per il dataset da 5 universita:

- RedSIB BDB: 339 MB
- RedSIB Virtuoso: X
- OSGi: 82 MB

Come ci si poteva aspettare, le dimensioni dello storage nel caso di RedSIBBDB è notevolmente maggiore rispetto ad OSGi. Comportamento inverso rispetto ai tempi di caricamento. OSGi impiega quindi un tempo maggiore ad occupare una dimensione su disco minore e viceversa avviene per RedSIB BDB. Questi due parametri non sono comunque comparabili. L'organizzazione sul disco è certamente più performante per OSGi.

5.2.2.4 Considerazioni finali

Nonostante il tempo di caricamento sia poco performante, OSGi recupera in termini di tempo di risposta delle query, per le quali restituisce sempre un risultato.Se volessimo fare una valutazione globale come prestazioni complessive OSGi resta comunque la più affidabile.Segue poi la RedSIBBDB la quale oltre ad essere la più rapida nel caricamento del dataset non è da meno nelle query. Gli unici limiti che si pone sono nel fallimento delle query Q1 Q2 Q3 e Q11 (per il dataset da 5 inuversità).Infine per la RedSIBVirtuoso è valida una considerazione contraria. È minore il numero di fallimenti, ma le tempistiche non sono le migliori.

5.2.3 Influenza dell'ordine dei pattern sull'esecuzione delle queries

Giunti al termine dei test abbiamo provato, supponendo di conoscere la base di conoscenza, a girare l'ordine dei "triple patterns" delle query. I risultati associati ad ogni triple pattern, infatti, sono diversi. Mettendo prima quelli che forniscono meno risultati, siamo riusciti ad ottenere esiti positivi per tutte le query del dataset da 1 università e per tutte le query ad eccezione della query Q2 del dataset da 5 università. Esistono numerosi studi al riguardo ed in uno di questi (fondato proprio sul LUBM) lo possiamo ritrovare nell'articolo: " SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation" di Stocker et [42].

Di seguito sono riportate le query modificate:

QUERY 1:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE {?X ub:takesCourse
<http://www.department0.university0.edu/GraduateCourse53> .
?X rdf:type ub:GraduateStudent}
```

QUERY 2:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
```

```
SELECT ?X ?Y ?Z
WHERE
{?X ub:undergraduateDegreeFrom ?Y .
?Z ub:subOrganizationOf ?Y.
?Y rdf:type ub:University.
?Z rdf:type ub:Department.
?X rdf:type ub:GraduateStudent.
?X ub:memberOf ?Z}
```

QUERY 3:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE
{?X ub:publicationAuthor <http://www.Department0.University0.edu/AssistantProfessor0> .
?X rdf:type ub:Publication}
```

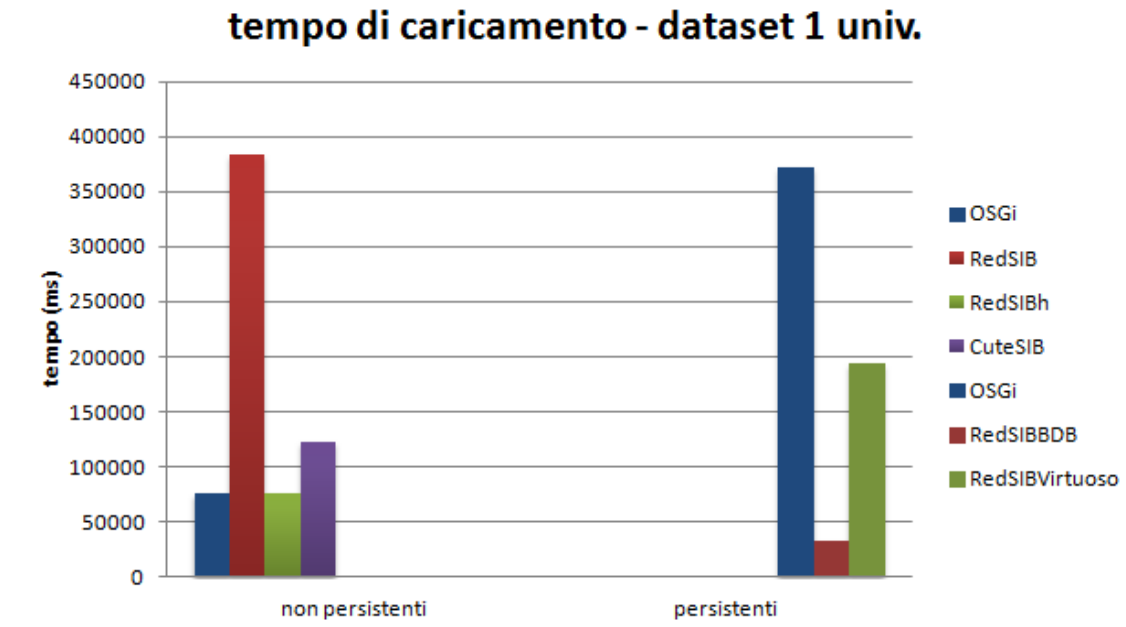
QUERY 11:

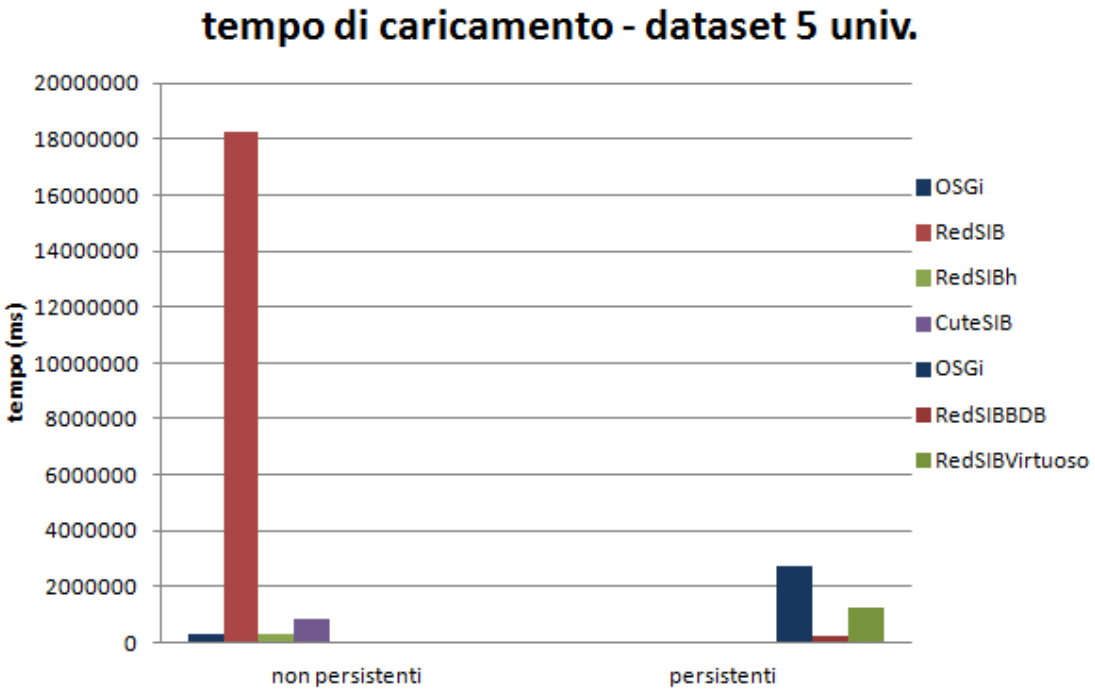
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE
{ ?X ub:subOrganizationOf <http://www.University0.edu> .
{?X rdf:type ub:ResearchGroup}
```

Ovviamente ci siamo limitati a fare questa operazione per le sole query che avevano riportato un globale fallimento. Deducendo che per RedSIB e le SIB su essa basate, questa modifica è deleteria. La causa di questo comportamento è

certamente un motore SPARQL non di ultima generazione. Tuttavia non è corretto considerare i dati temporali ottenuti con questa nuova configurazione come validi, in quanto derivanti da una modifica non prevista nel test originale

5.2.4 Analisi dei tempi di caricamento





Una struttura dati persistente è una struttura dati memorizzata su una memoria persistente (permanente), come può essere un disco rigido. Viceversa la forma non persistente o volatile, seppur più veloce, necessita dell'alimentazione elettrica al fine di mantenere memorizzate le informazioni, ne è un esempio la RAM.

Da una analisi schematica dei tempi di caricamento del dataset nei due differenti ambienti di test, si evincono ulteriori risultati:

si può notare l'evidente perdita di performance di OSGi nel caricamento dello stesso numero di triple dal caso non persistente a quello persistente; e viceversa la rapidità di RedSIB nelle condizioni di caricamento su RAM.

Discorso analogo può essere fatto in merito al test delle 5 università, ma se volessimo essere più precisi e confrontare queste due SIB nella totalità, possiamo notare come all'aumentare delle triple, nel caso persistente, la perdita di velocità di OSGi si fa sempre meno significativa, realtà non vera per il caso non persistente dove il dislivello fra RedSIB e OSGi non è trascurabile.

Per le restanti SIB non è corretto fare un confronto in quanto si tratta di implementazioni differenti.

5.2.5 Analisi sulla completezza delle queries

Per la valutazione della completezza e solidità delle query ci siamo rifatti nuovamente all'articolo LUBM, dove è riportato il numero totale di risposte per ogni singola query, anche se dedicato al solo dataset da 1 università. In particolare si tratta di una tabella riassuntiva dei risultati raccolti dal test LUBM originale, ma ci è stata ugualmente utile come confronto.

Query soundness of OWLJessKB

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Correct answers/ total answers	4/4	0/0	6/6	34/41	719/719	7790/8330	67/67	7790/8330	208/208	4/4	224/224	15/540	1/1	5916/5916
Soundness	100	100	100	83	100	94	100	94	100	100	100	3	100	100

Si può dedurre che per le query prive di reasoning come la Q1 Q2 Q3 e Q14, il numero totale di risposte è quello atteso. Diverso è per le restanti query, che destinate a una valutazione di reasoning, nel nostro contesto, non sono state considerate.

6 Estensione LUBM per Publish/Subscribe

Fatte queste considerazioni in merito ad un insieme di test mirati a descrivere le prestazioni in un contesto di query, risulta interessante, una volta compreso a pieno il benchmark LUBM, pensare ad una sua possibile estensione al contesto di publish/subscribe. Ulteriore obiettivo di questa tesi è fornire un punto di partenza per eventuali test futuri, dando a questi meri numeri un significato interessante e pratico, con l'intenzione di migliorare l'impiego futuro della piattaforma semantica Smart-M3.

La sottoscrizione può essere vista come una query alla quale il Subscriber si sottoscrive con lo scopo di ricevere una notifica, quando la risposta relativa a tale query, muta nel tempo.

Pensando ad una applicazione del dataset offerto dal benchmark LUBM, dobbiamo immaginare un ambiente universitario descritto da una logica semantica e memorizzato sottoforma di grafi. Fino a questo punto nulla è invariato, ma possiamo già iniziare a fare alcune considerazioni. Ricordando le classi del dataset suddivise in "organizzazioni", "persone", etc. possiamo immaginare un reale bisogno di sapere se un professore è cambiato o uno studente si è ritirato, senza bisogno di effettuare di nostra iniziativa la query in grado di fornirci le risposte desiderate.

Secondo il contesto classico di P/S i messaggi scambiati tra KP e SIB contengono dati e le sottoscrizioni altro non sono che vincoli a tali dati. Può essere importante valutare, nell'area delle prestazioni, i parametri di "reattività" , ovvero la capacità di fornire una notifica e la "velocità del processo", indice di completezza.

Al primo parametro potremmo far corrispondere un valore temporale "tempo di notifica", che tenga conto dell'intervallo che intercorre fra la creazione di una richiesta (update da parte di un KP) al feedback di notifica fornito a quest'ultimo (risposta inviata dalla SIB al KP).

La seconda metrica, non meno importante, potrebbe essere il "rendimento" , indice del tasso di richieste che un sistema può elaborare correttamente entro un determinato intervallo di tempo. Ovvero parametro che definisce il numero di pubblicazioni richieste, che vengono completamente elaborate nello stesso intervallo di tempo, in cui sono state generate le richieste.

REVISIONE DELLE QUERY:

Giunti a questo punto possiamo procedere con una reinterpretazione delle query già visionate in precedenza, pensando ad un reale caso di sottoscrizioni:

La query 1 ad esempio seleziona i “graduate student” che hanno scelto il corso <http://www.department0.university0.edu/GraduateCourse53>. In questo contesto, potremmo ad esempio aver bisogno di ricevere una notifica nel caso in cui cambino i “graduate student” per quel particolare corso.

Ancora, nella query 2 vengono selezioni i “Graduate Student” membri del “University department” di una specifica “University” e iscritti ad un corso di laurea nella medesima “University”. Le informazioni riportate dalla query sono tre: “Graduate Student”, “University” e “Department”.

Una possibile sottoscrizione si può verificare quando un nuovo studente, che soddisfa i requisiti, è aggiunto o rimosso dalla lista, o ancora quando vengono modificati i dipartimenti di un’università o addirittura aggiunto o rimosso un intero complesso universitario. Certo le ultime due realtà sono meno frequenti e probabili, ma ugualmente segnalabili.

La query 3 riporta le pubblicazioni il cui autore appartiene al <http://www.Department0.University0.edu/AssistantProfessor0>. Sottoscriversi a questa query significherebbe riceve una notifica nel caso in cui venisse aggiunta una nuova pubblicazione.

Segue la query 4 nella quale vengono selezionati i professori che lavorano per uno specifico dipartimento e ne riporta tra i risultati anche nome, indirizzo e numero di telefono. Sottoscriversi a questa query consente di ottenere una segnalazione nel caso in cui un professore cambiasse indirizzo, numero di telefono o addirittura dipartimento.

Nella query 5 vengono individuate le persone che sono membri del dipartimento <://www.Department0.University0.edu>. Una sottoscrizione a questa query

significherebbe ricevere una notifica per ogni persona aggiunta a quello specifico dipartimento.

La query 6 riporta tutti gli studenti. Un semplice caso di notifica si avrebbe con l'aggiunta di nuovi studenti.

La query 7 è simile alla 6, ma riporta la lista di studenti iscritti al corso in cui il professore <http://www.Department0.University0.edu/AssociateProfessor0> insegna. In particolare la query fornisce come risposta gli "student" e i "course" interessati. Una sottoscrizione a questa query comporterebbe una notifica nel caso di aggiunta di uno studente o di modifica del corso di insegnamento svolto dallo stesso professore.

Nella query 8 vengono selezionati gli studenti membri dei dipartimenti facenti parte dell'università <http://www.University0.edu>. Come risposta vengono indicati "Student", "emailAddress" dello studente e "Departemnet".

Una possibile sottoscrizione a tale query porterebbe ad una segnalazione nel caso in cui aumentasse il numero di studenti o più semplicemente nel caso in cui uno di questi variasse il suo indirizzo e-mail.

La query 9 riporta "Student", "Faculty" e "Course". Seleziona gli studenti frequentanti un certo corso e i consulenti delle facoltà che possiedono lo stesso corso di insegnamento. Se si aggiungessero nuovi studenti, che rispecchiassero ancora questi connotati, allora la sottoscrizione a questa query comporterebbe una notifica. Ancora, potrebbero essere aggiunti dei corsi alle facoltà e quindi ulteriormente incrementati il numero di studenti che soddisfano questa query.

La query 10 riportando gli studenti frequentati il corso <http://www.Department0.University0.edu/GraduateCourse0>, potrebbe essere oggetto di sottoscrizione nel semplice caso in cui aumenti il numero di studenti.

La query 11 è altrettanto semplice, riporta la lista dei gruppi di ricerca dell'organizzazione <http://www.University0.edu> . Dunque in caso di

eliminazione o supplemento di un “Research Group” questo verrebbe immediatamente segnalato.

Segue la query 12 la quale restituisce le “Chair” utilizzate dai dipartimenti dell’università <http://www.University0.edu>. L’incremento dei dipartimenti o semplicemente l’aumento di sedie utilizzate nell’università individuata implicherebbe una notifica al client sottoscritto a questa query.

La query 13 seleziona le persone riconosciute come alunni dell’università <http://www.University0.edu>. Una sottoscrizione a questa query rivelerebbe una notifica nel caso in cui qualche persona e nello specifico alunni si iscrivessero in tale università.

Infine la query 14 riportando tutti gli “UndergraduateStudent”, invierebbe una notifica ai subscriber a partire dal semplice caso di incremento di tale classe.

7 Conclusioni e sviluppi futuri

Da una analisi complessiva dei risultati ottenuti dall'esecuzione del benchmark LUBM, OSGi risulta essere la piattaforma semantica di Smart-M3 più affidabile ed anche più performante nella forma non persistente. Proiettandoci in un prossimo futuro questa SIB sarebbe certamente la prima scelta nelle applicazioni reali, anche come persistente, in quanto rapida e sicura. In ambito Biomedicale tuttavia, la solidità nella memorizzazione dei dati è essenziale. In molti casi la tracciabilità dei parametri vitali di un paziente permette di trarne una diagnosi. La perdita di questi dati comporterebbe quindi disastrose conseguenze.

In merito alla nativa CuteSIB ancora non possiamo evidenziare rilevanti caratteristiche. Necessita ancora di un ulteriore crescita.

RedSIB infine allo stato attuale esige delle migliorie. Dunque basandosi sul set di risultati ottenuti è necessario investigarne le fondamenta e possibilmente risolverne le cause.

Terminato questo lavoro di tesi è nel mio personale interesse sottolineare non si tratti necessariamente di un lavoro concluso, ma ispirazione e punto di partenza per possibili approfondimenti futuri. Ne possono essere un esempio l'esecuzione di altri benchmark, come ad esempio "The Berlin SPARQL Benchmark" sul completo set di SIB o "SP2Bench: A SPARQL Performance Benchmark" già eseguito per le diverse implementazioni di RedSIB e OSGi, ma non per la nativa CuteSIB.

Con riferimento alla sezione 5 di questo elaborato, spero di aver suscitato un significativo interesse nella volontà di estendere il benchmark LUBM anche per Publish/Subscribe. Ambiente di utilizzo non lontano da una realtà applicativa, anche in ambito biomedico. Un nuovo benchmark di questa portata risulterebbe essenziale per testare sistemi verso l'innovazione.

Infine note le prestazioni di OSGi (non persistente) e le sue carenze in ambito persistente, sarebbe opportuno risolverne i limiti ai fini di renderla completa.

Promuovendo la sua direzione di sviluppo.

8 Appendice

8.1 Codice SPARQL delle query del benchmark LUBM

Tutte le query sotto elencate condividono gli stessi prefissi:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
```

QUERY 1

Questa query prevede in ingresso una grande quantità di dati, limitati ad una sola classe e vuole riportare una risposta altamente selettiva, senza però fornire alcuna informazione gerarchica o deduttiva.

```
SELECT ?X
WHERE {?X rdf:type ub:GraduateStudent.
?X ub:takesCourse <http://www.department0.university0.edu/GraduateCourse53>}
```

QUERY 2

Questa query aumenta di complessità. Sono infatti incluse tre classi e due proprietà. Inoltre vi è uno schema triangolare di relazione tra gli oggetti coinvolti.

```
SELECT ?X ?Y ?Z
WHERE
{?X rdf:type ub:GraduateStudent.
?Y rdf:type ub:University.
?Z rdf:type ub:Department.
?X ub:memberOf ?Z.
?Z ub:subOrganizationOf ?Y.
?X ub:undergraduateDegreeFrom ?Y}
```

QUERY 3

Questa query è simile a Query 1 ma la classe “pubblication” ha una vasta gerarchia.

```
SELECT ?X
WHERE
{?X rdf:type ub:Publication.
?X ub:publicationAuthor
 <http://www.Department0.University0.edu/AssistantProfessor0>}
```

QUERY 4

Questa query ha un ingresso limitato e alta selettività. Assume la “subClassOf” tra “professor”, che ha una vasta gerarchia e le sue “Subclasses”. Inoltre questa query si interroga su proprietà multiple di una singola classe.

```
SELECT ?X ?Y1 ?Y2 ?Y3
WHERE
{?X rdf:type ub:Professor.
?X ub:worksFor <http://www.Department0.University0.edu>.
?X ub:name ?Y1.
?X ub:emailAddress ?Y2.
?X ub:telephone ?Y3}
```

QUERY 5

Questa query presuppone la relazione “subClassOf” tra la “person” e le sue “subcalsses” e la relazione “subPropertyOf” tra “memberOf” e le sue “subproperties”. Inoltre la classe persona presenta una gerarchia molto ampia.

```
SELECT ?X
WHERE
{?X rdf:type ub:Person.
?X ub:memberOf <http://www.Department0.University0.edu>}
```

QUERY 6

Questa query interroga su una sola classe. Assume una relazione “subClassOf” esplicitata tra il “UndergraduateStudent” and “Student” e una relazione implicita tra “GraduateStudent” e “Student”. Inoltre, dispone di ampio ingresso e bassa selettività.

```
SELECT ?X
WHERE
{?X rdf:type ub:Student}
```

QUERY 7

Questa query è simile a Query 6 in termini di classe “Student”, ma aumenta il numero di classi e proprietà e la sua selettività è più elevata.

```
SELECT ?X ?Y
WHERE
{?X rdf:type ub:Student.
?Y rdf:type ub:Course.
?X ub:takesCourse ?Y.
<http://www.Department0.University0.edu/AssociateProfessor0>,
ub:teacherOf, ?Y}
```

QUERY 8

Questa query è ulteriormente più complessa della Query 7. Prevede infatti l’aggiunta di una un’ulteriore variabile di ricerca.

```
SELECT ?X ?Y ?Z
WHERE
{?X rdf:type ub:Student.
?Y rdf:type ub:Department.
?X ub:memberOf ?Y.
?Y ub:subOrganizationOf <http://www.University0.edu>.
?X ub:emailAddress ?Z}
```

QUERY 9

Oltre alle caratteristiche della classe “Student” e l’ampia gerarchia della classe “Facolty”, come già visto nella Query 2, questa query è caratterizzata dai più classi e proprietà nel set di query e possiede uno schema triangolare di relazioni.

```
SELECT ?X ?Y ?Z
WHERE
{?X rdf:type ub:Student.
?Y rdf:type ub:Faculty.
?Z rdf:type ub:Course.
?X ub:advisor ?Y.
?Y ub:teacherOf ?Z.
?X ub:takesCourse ?Z}
```

QUERY 10

Questa query differisce dalle Query 6-9, in quanto richiede solo la relazione implicita "subClassOf" tra "GraduateStudent" e "Student", ovvero la relazione "subClassOf" tra il "UndergraduateStudent" e "Student" non aggiunge altri risultati.

```
SELECT ?X
WHERE
{?X rdf:type ub:Student.
?X ub:takesCourse
 <http://www.Department0.University0.edu/GraduateCourse0>}
```

QUERY 11

Le Query 11-13 hanno lo scopo di verificare la presenza di alcune capacità di reasoning nel sistema OWL. In questa query, la proprietà è "subOrganizationOf" è definita come transitiva. I dati di riferimento, ovvero le istanze di "ResearchGroup" sono iscritte sotto "organization" di un Dipartimento e la successiva "subOrganization" di un individuo University. Richiedono per la risposta l'inferenza tra la "subOrganizationOf", le istanze di "ResearchGroup" e "University". Inoltre, il suo ingresso è piccolo.

```
SELECT ?X
WHERE
{?X rdf:type ub:ResearchGroup.
?X ub:subOrganizationOf  <http://www.University0.edu>}
```

QUERY 12

I dati di riferimento non producono alcuna istanza di classe Chair sotto “person”, ma di classe Chair sotto “Professor”. Quindi, questa query richiede l'inferenza che “professor” è un’istanza della classe “Chair” perché lui o lei è a capo di un dipartimento. L’ingresso di questa query è piccolo.

```
SELECT ?X, ?Y
WHERE
{?X rdf:type ub:Chair.
?Y rdf:type ub:Department.
?X ub:worksFor ?Y.
?Y ub:subOrganizationOf. <http://www.University0.edu>}
```

QUERY 13

La proprietà “hasAlumnus” è definita nell'ontologia come l'inverso della proprietà “degreeFrom”, che ha tre sottoproprietà: “undergraduateDegreeFrom”,“mastersDegreeFrom” e “doctoralDegreeFrom”. I dati di riferimento identificano una persona come un alunno di una università utilizzando uno di questi tre sottoproprietà anziché hasAlumnus. Pertanto, questa query assume la relazione “subPropertyOf” tra “degreeFrom” e le sue sottoproprietà e richiede anche l'inferenza su “inverseOf”.

```
SELECT ?X
WHERE
{?X rdf:type ub:Person.
<http://www.University0.edu> ub:hasAlumnus ?X}
```

QUERY 14

Questa query è la più semplice nel set di test. Questa query rappresenta un ampio ingresso e bassa selettività senza ottenere alcuna informazione gerarchica o deduttiva.

```
SELECT ?X
WHERE
{?X rdf:type ub:UndergraduateStudent}
```

9 Bibliografia

Internet of Things

[1] Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." /Computer networks/ 54.15 (2010): 2787-2805.

[2] Perera, Charith, et al. "Context aware computing for the internet of things: A survey." /Communications Surveys & Tutorials, IEEE/ 16.1 (2014): 414-454.

Telemedicina

[3] http://www.telemedware.com/telemedicina/telemedicina-in-italia.aspx

[4] Manzi, Elisa, Silvia Selvaggi, and Vincenzo Sica. "Tecnologie informatiche e delle comunicazioni in medicina: la telemedicina." Telemedicina. Springer Milan, 2010. 1-9.

[5] Vergari, Fabio, et al. "An integrated framework to achieve interoperability in person-centric health management." International journal of telemedicine and applications 2011 (2011): 5.

Interoperabilità

[6]http://www.treccani.it/enciclopedia/interoperabilita_(Enciclopedia_della_Scienza_e_della_Tecnica)/

[7]http://www.agid.gov.it/sites/default/files/documentazione_trasparenza/cdc-spc-gdl6-interoperabilitasemopendata_v2.0_0.pdf

Web Semantico

[8] Barnaghi, Payam, et al. "Semantics for the Internet of Things: early progress and back to the future."/International Journal on Semantic Web and Information Systems (IJSWIS)/ 8.1 (2012): 1-21.

[9] Hendler, James, Ora Lassila, and T. Berners-Lee. "The semantic web." /Scientific American/ 284.5 (2001): 34-43.

[10] Berners-Lee, Tim, James Hendler, and Ora Lassila. "The semantic web." /Scientific american/ 284.5 (2001): 28-37.

[11] Matthews, Brian. "Semantic web technologies." /E-learning/ 6.6 (2005): 8.

[12] Gómez-Pérez, Asunción, and Oscar Corcho. "Ontology languages for the semantic web." /Intelligent Systems, IEEE/ 17.1 (2002): 54-60.

URI

[13]Berners-Lee, Tim, Roy Fielding, and Larry Masinter. *Uniform resource identifier (URI): Generic syntax*. No. RFC 3986. 2004.

XML

[14]ray, Tim, et al. "Extensible markup language (XML)." /World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210/ 16 (1998).

[15]http://www.dima.unige.it/~pascarel/corso/slides/XML.pdf

RDF

[16]http://www.w3.org/RDF/

[17]Miller, Eric. "An introduction to the resource description framework." *Bulletin of the American Society for Information Science and Technology* 25.1 (1998): 15-19.

[18]http://www.w3c.it/papers/RDF.pdf

[19]http://www.ce.unipr.it/people/bianchi/Teaching/IntelligenzaArtificiale/WebSemantico_Ontologie/ResourceDescriptionFramework.pdf

Ontologie

[20]http://www.w3.org/TR/2012/REC-owl2-primer-20121211/

[21]McGuinness, Deborah L., and Frank Van Harmelen. "OWL web ontology language overview." /W3C recommendation/ 10.10 (2004): 2004.

Progetto Sofia

[22]Korzun, Dmitry G., et al. "Overview of Smart-M3 principles for application development." Proc. Congress on Information Systems and Technologies (IS&IT'11), Conf. Artificial Intelligence and Systems (AIS'11). Vol. 4. 2011.Nokia

[23]http://www.sofia-project.eu/

Smart-M3

[24]Honkola, Jukka, et al. "Smart-M3 information sharing platform." *The IEEE symposium on Computers and Communications*. IEEE, 2010.

[25]http://redis.io/topics/pubsub

[26]http://amslaurea.unibo.it/3726/1/emanuele_montemurro_tesi.pdf

Linguaggio SPARQL

[27]Pérez, Jorge, Marcelo Arenas, and Claudio Gutierrez. "Semantics and Complexity of SPARQL." /International semantic web conference/. Vol. 4273. 2006.

[28]http://www.w3.org/TR/rdf-sparql-query/

[29]http://www.w3.org/TR/rdf-sparql-XMLres/

[30]http://www.w3.org/TR/rdf-sparql-protocol/

RedSIB

[31]Schmidt, Michael, et al. "SP^ 2Bench: a SPARQL performance benchmark."*Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. IEEE, 2009.

[32]http://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/bristol-1.html

[33]http://librdf.org/

[34]http://www.ingegneriainformatica.unina.it/sites/default/files/elaborati_tesi/2015/03/Elaborato%20Losco%20Alessandro%20N46-874.pdf

[35]Olson, Michael A., Keith Bostic, and Margo I. Seltzer. "Berkeley DB." /USENIX Annual Technical Conference, FREENIX Track/. 1999.

[36]http://sourceforge.net/p/virtuoso/mailman/virtuosousers/thread/1369128859.27972.1385.camel@octo.iv.dev.null/

CuteSIB

[37]http://sourceforge.net/projects/smart-m3/files/CuteSIB_0.1.0/cutesib-0.1.0.tar.gz/download

OSGi

[38]Manzaroli, Daniele, et al. "Smart-M3 and OSGi: The interoperability platform." /Computers and Communications (ISCC), 2010 IEEE Symposium on/. IEEE, 2010.

[39]www.osgi.org

[40]http://www.programmazione.it/index.php?entity=eitem&idItem=39127

Benchmark

[41]Guo, Yuanbo, Zhengxiang Pan, and Jeff Heflin. "LUBM: A benchmark for OWL knowledge base systems." *Web Semantics: Science, Services and Agents on the World Wide Web* 3.2 (2005): 158-182.

[42]Stocker, Markus, et al. "SPARQL basic graph pattern optimization using selectivity estimation." *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008.

RINGRAZIAMENTI

Desidero ricordare tutti coloro che mi hanno aiutata e sostenuta nella stesura della tesi, con suggerimenti, critiche ed osservazioni, ma anche con una semplice risata o abbraccio: a loro va la mia gratitudine.

Ringrazio innanzitutto il professore Luca Roffia, docente che a partire dal suo insegnamento, fino alla elaborazione della tesi, mi ha motivata, trasmettendomi con il suo carisma la stessa gioia e lo stesso interesse che egli nutre verso la sua disciplina, diradando la preoccupazione sulle mie carenti conoscenze informatiche.

Sono grata agli ing. Fabio Viola e prof. Alfredo D'Elia che mi hanno accompagnata, supportata e sopportata rendendomi il lavoro meno ostico e sempre più spesso piacevole. Sempre disponibili e cordiali nel rispondere ai miei mille dubbi e preoccupazioni, certamente senza il loro contributo questa tesi non avrebbe preso forma.

Un sentito "Grazie" va alla mia famiglia.

Mio padre Luca Bertarelli e mia madre Maria Rosa Graldi, mio esempio e mio cuore. Da sempre ho visto in loro due lavoratori giusti e leali, sempre pronti a lottare. Loro sono la mia determinazione e la mia tenacia.

Mi hanno insegnato il valore della fatica e la gioia del risultato meritato, sempre mi hanno trasmesso il loro grande amore. Oltre ad essere i miei migliori sponsor a fondo perduto, sono sempre stati miei fan e spero di renderli tanto orgogliosi di me quanto io lo sono di loro (mamma non piangere!).

Un ringraziamento particolare va a Camilla Urbani e Federica Pagani, due amiche uniche e insostituibili, che mi hanno affiancata in questo lungo viaggio. Sono state il mio sostegno e la mia forza. Con loro lo studio è diventata una piacevole avventura e le sconfitte meno dolorose.

Mi hanno insegnato il giusto equilibrio fra piacere e dovere e resa parte di una vera amicizia destinata a non esaurirsi nel tempo.

Ringrazio tutti i miei numerosi famigliari, i miei nonni, zii e cugini, in particolare Chiara Graldi, Filippo Mancini, Francesca Graldi, Martina Mancini e Monica Mancini, sempre presenti nonostante la distanza, hanno perdonato e giustificato le mie assenze. Non sono mai mancati un consiglio o una erudita spiegazione, una corsa, una serata e una risata liberatoria. Loro sono senza dubbio il mio passato, presente e futuro e di questo sono immensamente grata.

Ancora rendo grazie ai miei compagni universitari, amici e sostenitori, grazie a loro le lunghe ore di lezione sono state tutto meno che noiose.

Sono riconoscente alla mia carissima amica e co-inquilina AnnaGiulia Stagni, la quale è stata testimone di tutte le mie alternanze umorali, senza di lei avrei passato molte notti insonni e sempre grazie a lei ne ho passate altrettante all'insegna dell'introspezione.

Infine desidero dire Grazie a tutti coloro che vicini o lontani mi hanno sempre pensata e hanno tifato per me.