ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
DIPARTIMENTO DI INGEGNERIA DELL'ENERGIA
ELETTRICA E DELL'INFORMAZIONE
"GUGLIELMO MARCONI"

CORSO DI LAUREA IN
INGEGNERIA ELETTRONICA PER L'ENERGIA E
L'INFORMAZIONE

Elaborazione ad alte prestazioni di eventi semantici: studio ed integrazione su piattaforma SEPA dell'algoritmo Rete

Elaborato in
Calcolatori Elettronici

Relatore

Prof. Ing. Luca Roffia

Correlatori

Ing. Gregorio Monari
Ing. Gianluca Di Tuccio

Presentato da

Marco Fontana

Anno Accademico 2022/2023

Alla mia famiglia

INDICE

ELENCO DELLE FIGURE

Figure

Grafici

Tabelle

ABSTRACT

Al giorno d'oggi le applicazioni software generano una grande quantità di dati, che aumenteranno esponenzialmente con il crescere delle nuove tecnologie di smart environment, legate all'Internet of Things, con l'avvento del web 3.0, che richiedono una elaborazione di dati su larga scala.

Con questa tesi si presenta l'integrazione di un algoritmo di notifica basato sull'algoritmo Rete all'interno della piattaforma semantica SEPA, per l'elaborazione di grandi quantità di eventi mediante standard RDF e SPARQL, ispirandosi al sistema INSTANS presentato in un articolo scientifico.

L'architettura SEPA è un software per lo sviluppo di applicazioni e servizi che permette di monitorare e notificare l'aggiunta o la rimozione di dati da un archivio, lo scopo di questa tesi è quello di studiare ed implementare un sistema per rendere più rapido il meccanismo di notifica di aggiornamenti per grandi quantità di dati.

INTRODUZIONE

Rete è un algoritmo ideato da Charles L. Forgy, informatico statunitense nato in Texas nel 1949, che verso la fine degli anni 70' pubblicò una tesi di dottorato in cui propose questo algoritmo di pattern-matching. Un algoritmo in grado di controllare la presenza di uno schema ricorrente, per il riconoscimento di corrispondenze all'interno di strutture dati. In questa tesi si discute l'applicazione di questo algoritmo per la piattaforma SEPA, applicazione che permette di notificare gli utenti riguardo l'aggiornamento di informazioni all'interno di database, lo scopo è quello di rendere più rapida l'esecuzione delle notifiche rispetto all'attuale algoritmo, soprattutto nel caso di grandi quantità di dati.

La discussione si apre con un'analisi del linguaggio di interrogazione SPARQL, per dati rappresentati in standard RDF.
In seguito, si descrive un sistema denominato INSTANS, per l'esecuzione di eventi tramite query SPARQL, mediante algoritmo Rete.
Questo algoritmo si compone di una propria struttura dati dalla forma di "albero ordinato", cioè composta da una serie di nodi, ciascuno dei quali contiene un pattern che collegato ad altri nodi forma una serie di pattern, composta da tutti i nodi precedenti, che possono rappresentare regole complete.
Per concludere, una analisi della complessità temporale, spaziale, ed una comparazione del tempo impiegato da SEPA a notificare i cambiamenti prima e dopo l'implementazione dell'algoritmo Rete.

1. TRATTAZIONE GENERALE

1.1 Database relazionali, RDF, SPARQL, SEPA

L' Internet delle Cose, in inglese Internet of Things (IoT) [1], fa riferimento all'estensione di internet a tutti gli oggetti, dove qualunque dispositivo elettronico è in grado di comunicare con altri attraverso l'interconnessione che lega tutti gli oggetti tra loro. Questo è possibile mediante la condivisione di informazioni mediante database [2], ai quali ogni dispositivo è in grado accedere per aggiungere o visualizzare informazioni in autonomia.

Un database è una collezione di informazioni inserite all'interno di un file che immagazzina grandi quantità informazioni nella memoria permanente, per effettuare un management, cioè permettere ad un utente di leggere o scrivere nel file, si utilizza un criterio denominato 'query language' [3], cioè un linguaggio di interrogazione che permette la lettura e scrittura di dati contenuti nel database.

In questa tesi si prende in considerazione lo SPARQL (*SPARQL Protocol and RDF Query Language*) [4], un linguaggio di interrogazione che può essere utilizzato in tecnologie per l'estrazione di informazioni di dati su database semantici [5], e mappati tramite RDF (*Resource Description Framework*) [6], modello per la descrizione delle informazioni che permette il riconoscimento di qualunque tipo di dato identificato tramite una sequenza di simboli che identifica univocamente una risorsa: URI (*Uniform Resource Identifier*) [7].

Secondo la logica dei predicati, le relazioni tra le informazioni sono facilmente esprimibili attraverso delle "asserzioni", componente chiave dell'RDF, costituita da tre elementi: "soggetto, predicato, oggetto", per questo anche detta "tripla". [8]

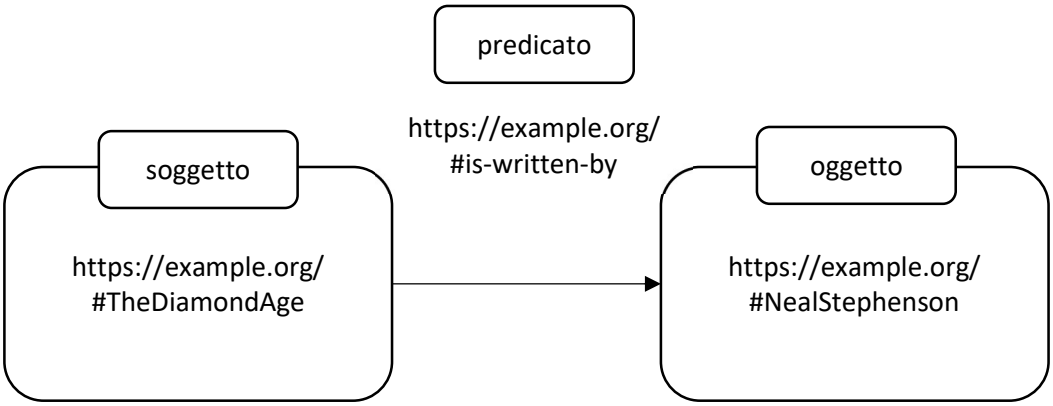

Figura 1 Esempio A di tripla RDF

Si possono formare relazioni più complesse se un singolo soggetto è in relazione con più oggetti, attraverso diversi predicati, ed altre relazioni possono prevedere l'utilizzo di un oggetto come soggetto di un'altra tripla a formare catene più complesse.

Possiamo immaginare queste risorse come immagazzinate in un database come una tabella, composta da tre colonne, che rappresentano soggetto, predicato e oggetto e che contengono in ogni riga una tripla completa:

Tabella 1 Esempio del contenuto di database relazionali

Soggetto	Predicato	Oggetto
Neuromancer	is-written-by	WilliamGibson
Neuromancer	is-a	Book
WilliamGibson	is-a	science-fiction-writer
MazeOfDeath	is-written-by	PhilipKDick
MazeOfDeath	is-a	book
PhilipKDick	is-a	science-fiction-writer

Per inserire o estrarre informazioni, è possibile effettuare un'interrogazione dell'archivio.
Se il nostro interesse è quello di visualizzare una parte dei contenuti all'interno del database, possiamo effettuare una lettura dei soli dati a cui siamo interessati, mediante una "query", cioè una domanda nella quale viene esplicitato il tipo di informazioni a cui siamo interessati, e ci vengono restituite tutte le triple del database che rilevano una corrispondenza tra essa e il contenuto del database. Mentre per eseguire l'aggiunta o rimozione di informazioni, si effettua un "update" di quest'ultimo.

La funzione di aggiunta e rimozione di triple dal database è al centro di questa tesi, perché SPARQL non prevede di mantenere una cronologia dei cambiamenti effettuati nell'archivio da inoltrare al client. Questo è proprio lo scopo di SEPA [9] che mediante un algoritmo, permette la trasmissione di notifiche che mostrano tutti i cambiamenti effettuati nel database, aggiungendo quindi un output non previsto dal linguaggio SPARQL.
Questo avviene mediante il meccanismo di sottoscrizione, una funzione che permette all'utente di segnare alcune specifiche query che, oltre a generare subito una risposta con i risultati del database, vengono eseguite ad ogni aggiornamento, e se questo prevede una modifica del database non ancora notificata, viene inoltrata verso il client.

Lo scopo di questa tesi è proprio quello di studiare e implementare un differente meccanismo di notifica, al fine di inoltrare in tempi più rapidi, i cambiamenti nel caso di grandi quantità di triple, ispirandosi al sistema chiamato INSTANS [10] che permette l'aggiunta di 'regole' per gestire determinate azioni nel caso di corrispondenza tra i dati.

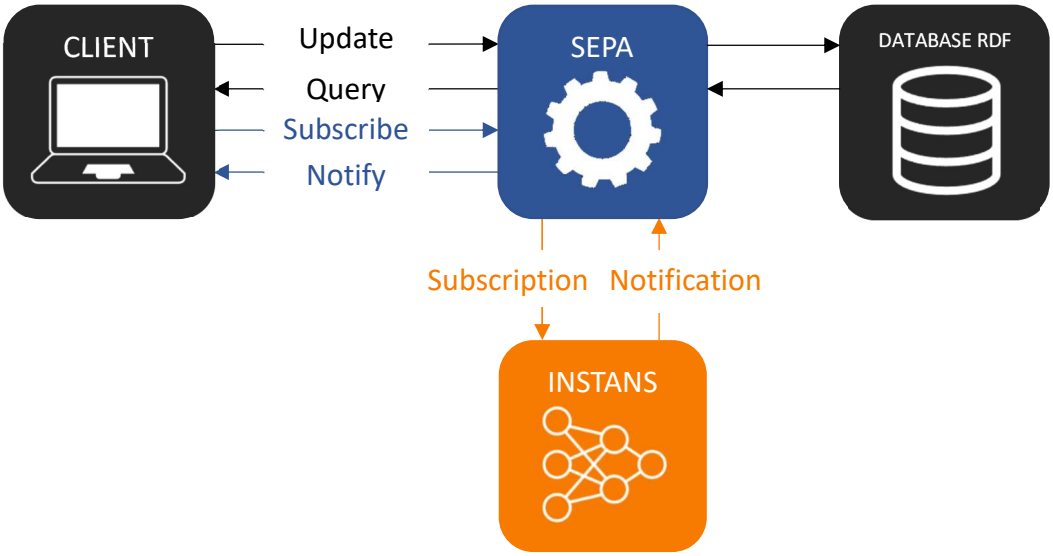

Figura 2 schema complessivo sistema SEPA e INSTANS

1.2 Sistema INSTANS e algoritmo Rete

INSTANS è un sistema che permette di utilizzare gli standard RDF e SPARQL per l'elaborazione di grandi quantità di dati, sotto forma di triple.
Il funzionamento generale è quello di trovare la relazione che compone le triple ed eseguire specifiche azioni nel caso di corrispondenze.

Prima di entrare nel dettaglio è necessario spiegare alcuni concetti fondamentali di intelligenza artificiale, essendo il sistema basato su un algoritmo di questo tipo.

L'intelligenza artificiale consente ad un sistema di prendere decisioni in autonomia. Un caso particolare sono i sistemi a regole [11] che utilizzano appunto delle "regole" cioè delle condizioni che se sono soddisfatte eseguono specifiche azioni, simulando lo stesso procedimento utilizzato dalla mente umana.

Le regole sono composte da due componenti, LHS (*Left Hand Side*) e RHS (*Right Hand Side*) che definiscono rispettivamente le "condizioni" da rispettare per eseguire determinate "azioni".
Analogamente ai linguaggi di programmazione, si può immaginare una regola come un costrutto 'if', che contiene le condizioni da rispettare, e una volta rispettate esegue l'azione contenuta nel corpo della funzione.

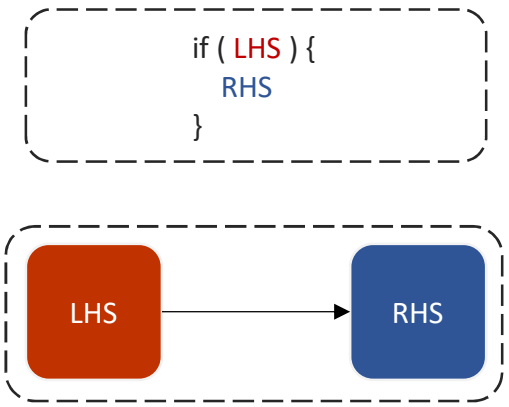

Figura 3 Analogia tra regole a singola condizione

Nel caso in cui siano presenti più condizioni, bisogna verificare che queste siano tutte rispettate prima di poter eseguire l'azione.
Dal punto di vista logico, si può schematizzare questa regola utilizzando un AND, che svolge per l'appunto la funzione di verifica che siano rispettate tutte le condizioni prima di eseguire l'azione.

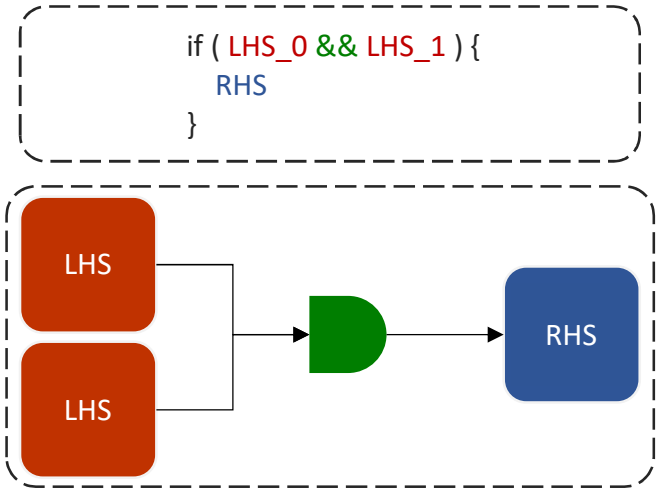

Figura 4 Analogia tra regole con condizioni multiple

Nel nostro caso, considerando il sistema INSTANS, le regole sono le 'iscrizioni' del SEPA, le condizioni (LHS), pattern di triple di cui è abilitato il meccanismo di notifica, e servono per verificare se l'aggiunta o rimozione di una determinata tripla riguarda le iscrizioni attive.

Nel caso in cui vengano soddisfatte tutte le condizioni, cioè sia rilevata una corrispondenza tra i dati in ingresso e una o più iscrizioni, verrà eseguita l'azione (RHS), che nel nostro caso sarà la notifica del cambiamento appena effettuato nel database.

Analogamente a quanto mostrato sopra, il vantaggio dell'algoritmo Rete [12] è quello di immagazzinare in memoria i risultati intermedi degli elementi che hanno rispettato le condizioni durante la ricerca di pattern.

Ogni oggetto di questo tipo prende il nome di 'nodo'.

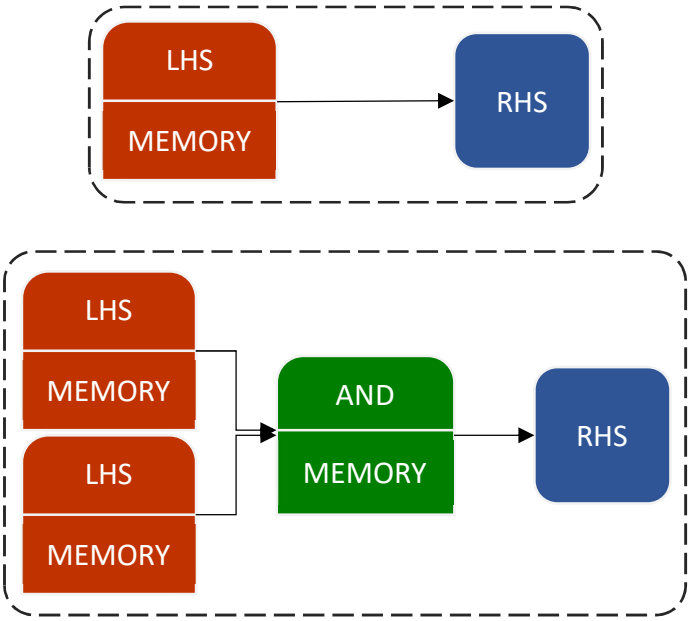

Figura 5 analogia tra regole Rete

Per iscrizioni semplici, composte da un unico pattern di triple, sono utilizzati un solo ‘LHS’ e ‘RHS’, quindi è sufficiente un unico nodo, mentre per query composte da più triple, dove è necessario effettuare un’operazione di ‘AND’ per controllare se tutti i pattern sono rispettati, si devono utilizzare più nodi in cascata, a formare una struttura ad albero.

I nodi iniziali, che contengono i singoli pattern di triple, sono chiamati “alpha", e sono i nodi di partenza da cui dirama la struttura. I nodi intermedi, che effettuano l’operazione di ‘AND’ logico, sono chiamati “beta", e il loro scopo è quello di propagare i risultati intermedi il cui pattern di tutti è stato rispettato.
La rete di nodi beta può essere evitata nel caso di regole con una sola condizione, infatti, in tal caso, è sufficiente un unico nodo alpha per eseguire il ‘RHS’ una volta rispettata la condizione. Quindi la struttura di nodi beta è opzionale e utilizzata solamente quando necessario.

L'azione è eseguita solo una volta attraversato l’ultimo nodo del ramo, il nodo contenente il ‘RHS’, per questa tesi è rappresentato da un nodo ‘SELECT’ che porterà in uscita le modifiche effettuate con l’ultimo update.

Nel caso in cui siano presenti più regole, è possibile utilizzare una singola struttura di nodi, nella quale regole che presentano pattern in comune possono essere riutilizzati per evitare ridondanza, formando strutture ad albero complesse nella quale alcuni nodi avranno più di un nodo figlio.

Questo permette di rendere indipendente Rete dal numero di regole, in quanto sono tutte contenute in una unica struttura complessiva che condivide le condizioni comuni.

Il sistema INSTANS permette di utilizzare questo algoritmo per gestire grandi quantità di dati a partire da query SPARQL, permettendo di riutilizzare parti di query equivalenti e salvare i risultati intermedi nella memoria dei nodi.

La struttura complessiva può assumere forme di questo genere:

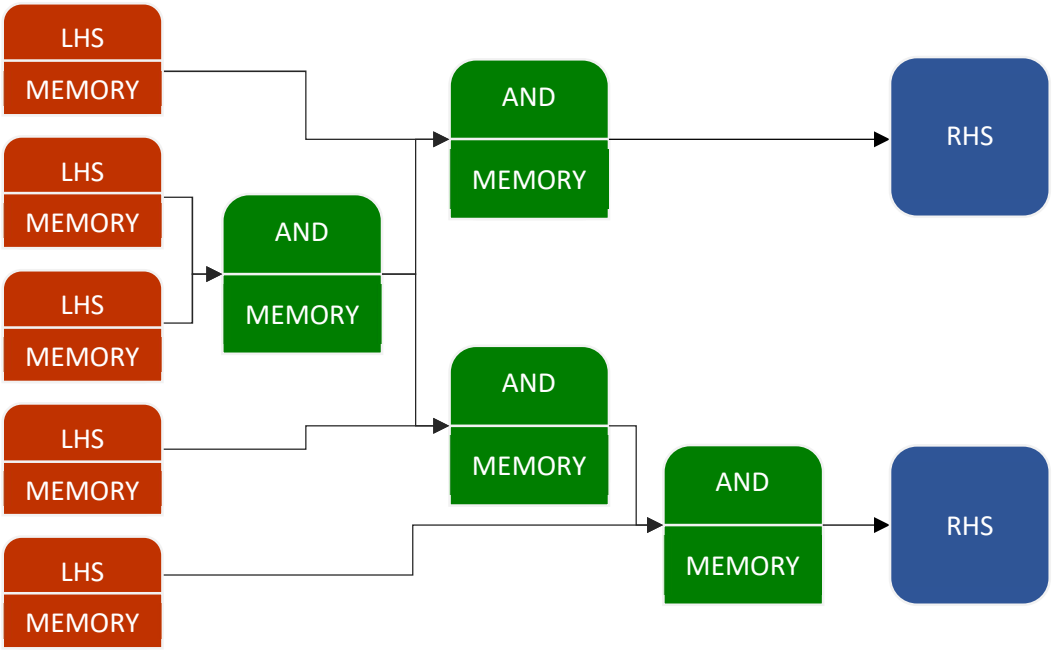

Figura 6 Esempio di struttura ad albero complessiva di Rete

Per effettuare la ricerca di corrispondenze tra le iscrizioni del SEPA, ogni tripla aggiunta o rimossa dal database deve essere comparata ad ogni condizione contenuta nei nodi alpha, se viene rilevato un pattern comune, cioè la tripla della query contenuta nel nodo ha pattern che corrisponde a quello della tripla aggiornata, viene propagato ulteriormente il valore della variabile corrispondente a quello della tripla.

Questa operazione, che prende il nome di “alpha matching", serve per selezionare i nodi alpha che hanno rilevato un aggiornamento in modo da poter propagare i cambiamenti verso i beta, e se vengono raggiunti gli ultimi nodi dei rispettivi rami, viene notificato l’aggiornamento.

2. ANALISI DELLE PRESTAZIONI

Per effettuare un'analisi delle prestazioni dell'algoritmo, lo vediamo applicato ad alcuni test in cui teniamo traccia della quantità di operazioni effettuate al crescere del numero degli ingressi.

Nello specifico i test sono stati effettuati su una macchina i7-8750 da 2.20GHz e 16GB di memoria RAM a partire da update e sottoscrizioni generate casualmente, e poi posizionate in ingresso all'algoritmo, dopo ogni esecuzione è stata effettuata la rilevazione necessaria a valutarne le prestazioni e i risultati sono stati riportati in alcuni grafici.
Di seguito un esempio di uno di essi:

Tabella 2 esempio di update e query utilizzati come test

UPDATE	QUERY

PREFIX ex:https://example.org/ex#
INSERT DATA {
ex:1 ex:2 ex:3 ;
ex:2 ex:5 .
ex:6 ex:4 ex:9
}
	SELECT * WHERE {
	ex:1 ?p1 ?o1 ;
	?p2 ex:5 .
	?s3 ?p3 ex:9
	}

Distinguiamo due tipi di complessità: complessità temporale e spaziale, rispettivamente il tempo impiegato a generare l'output in funzione della grandezza dell'input, e la quantità di memoria utilizzata durante l'esecuzione.

Nel nostro caso la complessità temporale fa riferimento al tempo impiegato per verificare la presenza di iscrizioni a cui notificare l'aggiornamento. L'analisi è effettuata considerando un numero di triple in ingresso crescente, cioè aggiornamenti composti da un numero di triple sempre maggiore.

La complessità spaziale indica il numero di elementi delle triple che hanno soddisfatto la condizione dei nodi e sono stati salvati al loro interno.

Prima di analizzare entrambe le complessità, vediamo alcuni limiti del sistema e le ottimizzazioni proposte da questa tesi per ridurne la complessità.

2.1 Limiti e ottimizzazioni

L'algoritmo contenuto nell'Alpha Matcher, se non ottimizzato, prevede che vengano controllati tutti nodi alpha della struttura, uno ad uno, dal primo all'ultimo, verificando quali di essi rilevino corrispondenza per ogni tripla in ingresso. Questa operazione ha una complessità temporale lineare in quanto non conoscendo in anticipo quali nodi rileveranno una corrispondenza, è necessario verificare il contenuto di ognuno di essi.

Per questa tesi si propone una semplice ottimizzazione che garantisce un tempo di esecuzione nettamente inferiore rispetto alla più semplice implementazione descritta in precedenza, che rende la complessità temporale logaritmica.
Eseguendo un'operazione di ordinamento dei nodi alpha ogni volta che viene aggiunto un nuovo nodo unico, secondo questo schema, si può modificare l'algoritmo di alpha-matcher e renderlo più efficiente:

- Inizialmente viene contato il numero di soggetti, predicati e oggetti presenti in tutti i nodi alpha della struttura. Poi vengono ordinati in base all'elemento che contiene il più alto di costanti, d'ora in poi chiamato "elemento RDF selezionato" (soggetto, predicato o oggetto):
- Nodi alpha che hanno nella condizione, come variabile, l'elemento RDF selezionato, vengono inseriti all'inizio dell'elenco.
- Nodi alpha che hanno nella condizione, come costante, l'elemento RDF selezionato, vengono inseriti dopo quelli che hanno rispettato il punto precedente e in ordine crescente secondo codifica ASCII.

Essendo i nodi ordinati, ad ogni aggiornamento del database, anziché controllare ogni nodo per ogni tripla contenuta nell'aggiornamento, verranno controllati

inizialmente tutti i nodi che contengono come pattern una tripla con l'elemento RDF selezionato variabile, mantenendo complessità lineare; una volta controllati tutti i nodi alpha di questo tipo, rimarranno da controllare solamente nodi con l'elemento RDF selezionato non rappresentato da variabili, a questo punto si può applicare un algoritmo di binary-search per trovare tutti i nodi che rilevano una corrispondenza con l'elemento RDF, e controllare unicamente questi nodi, rendendo la complessità temporale logaritmica.
In questo modo il numero effettivo di nodi attraversati sarà sempre minore o uguale a quello del caso precedente, garantendo un tempo di esecuzione inferiore, soprattutto per quanto riguarda grandi quantità di triple dell'aggiornamento.

Il limite dato da questa ottimizzazione è quello per cui la complessità può rimanere lineare nel caso in cui tutti i nodi dell'iscrizione contengano solamente variabili. Ma generalmente iscrizioni di questo tipo contengono un numero limitato di pattern di triple, quindi mantenendo molto ridotto il tempo di esecuzione.

2.2 Complessità temporale

Avendo eseguito diversi test, mantenendo un numero fisso di iscrizioni, l'algoritmo presenta una complessità temporale con andamento logaritmico in funzione del numero di triple dell'aggiornamento, questo andamento è dovuto al fatto che sia necessario controllare ogni nodo alpha, fino a quando necessario, per verificare quali di essi hanno pattern comune con la tripla in ingresso, a cui viene aggiunta la complessità dell'operazione logica dei nodi beta, che essendo molto ridotta rispetto a quella degli alpha, si può trascurare.

Effettuando vari test in cui ogni aggiornamento prevedeva l'inserimento di un numero crescente di triple da 1 a 't', generate casualmente a partire da una distribuzione uniforme, e un numero di iscrizioni costante (quindi un numero fissato di nodi), i risultati ottenuti sono stati riportati nel seguente grafico, dove ogni punto indica un aggiornamento composto da n_t triple:

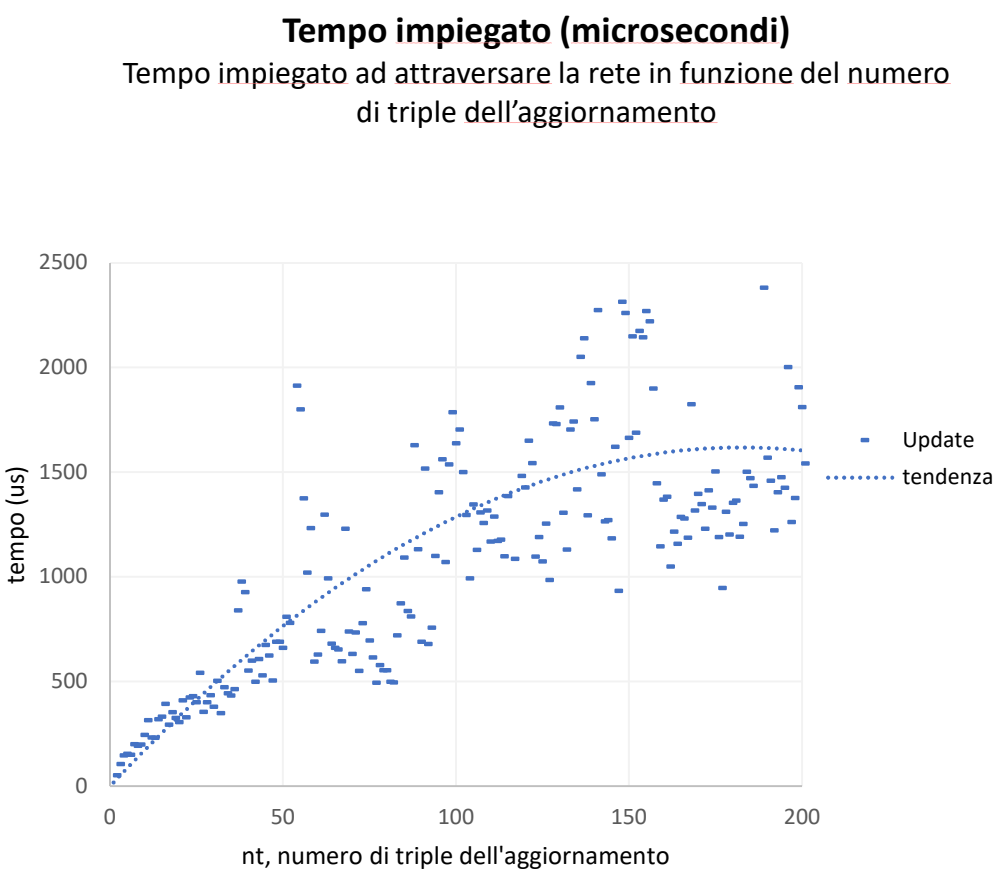

Grafico 1 Tempo (in microsecondi) in funzione del numero di triple

2.3 Complessità spaziale

Nel caso dell'algoritmo Rete, la complessità spaziale è molto elevata, con andamento esponenziale.
Questo è dovuto all'impostazione dell'algoritmo, è necessario effettuare un trade-off per mantenere una complessità temporale il più performante possibile, utilizzando una grande quantità di memoria.

Considerando l'applicazione dell'algoritmo al SEPA, questo vedrà al suo ingresso delle query SPARQL; quindi, nella memoria dei nodi verranno salvati i valori delle variabili che hanno rispettato il pattern delle triple.

Considerando il caso generale in cui verranno utilizzati sia i nodi alpha che beta, ogni nodo figlio conterrà al suo interno la memoria dei nodi genitore da propagare ulteriormente.
È facile intuire che l'andamento sarà esponenziale, in quanto ogni nodo figlio conterrà al suo interno la memoria complessiva di tutti i nodi genitori con andamento ricorsivo fino al raggiungimento dei nodi alpha iniziali.

Considerando gli stessi test effettuati in precedenza, cioè con un numero di triple dell'aggiornamento crescente e generate a partire da una distribuzione uniforme, riportando i risultati complessivi in un grafico dove le ascisse sono il numero di triple dell'aggiornamento e le ordinate il numero di elementi contenuti complessivamente all'interno della memoria dei nodi, otteniamo il seguente andamento:

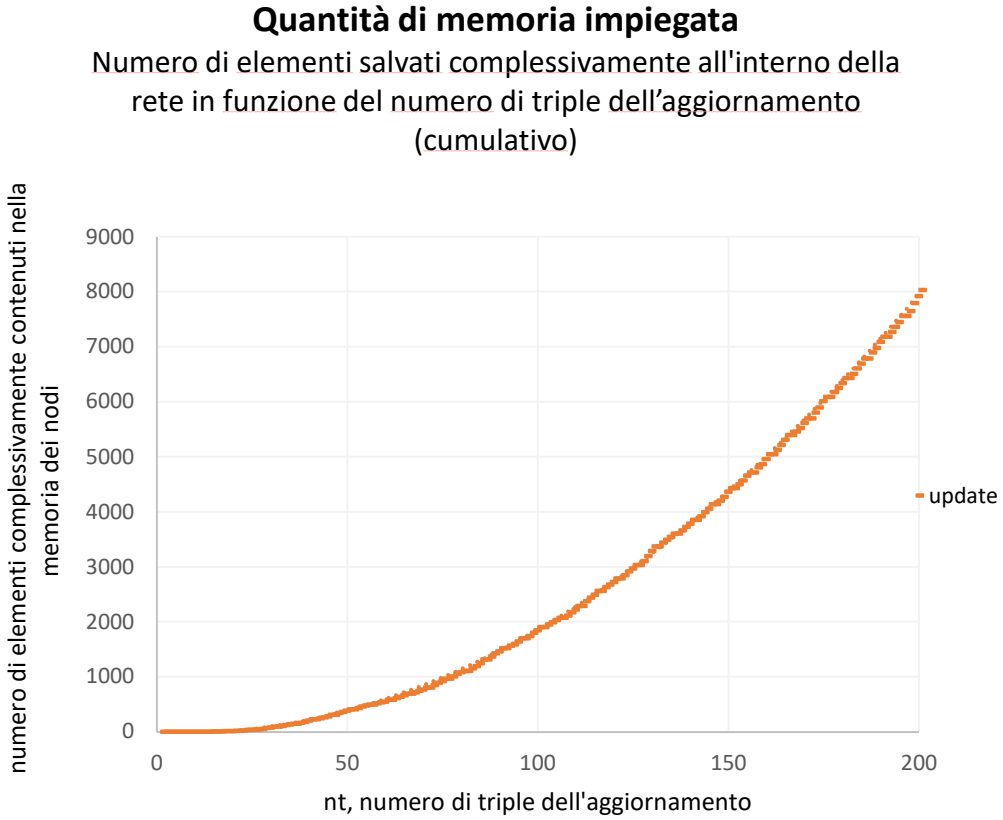

Grafico 2 Complessità spaziale complessiva

Per ogni aggiornamento composto da n_t triple, questo andamento è esponenziale in funzione del numero di triple dell'aggiornamento, la potenza a cui è elevato

dipende al numero di variabili contenute nella condizione dei nodi, questo perché ad ogni corrispondenza, ogni variabile aggiunge un elemento alla memoria del nodo che, se necessario, è propagata ulteriormente.

I risultati riportati nel grafico sono stati ottenuti a partire da iscrizioni dove ogni nodo conteneva minimo una variabile e massimo due, in modo da mantenere lo stesso valore medio tra numero di variabili e costanti contenuti nella condizione del nodo.

2.4 Comparazione con SPARQL

SEPA mediante algoritmo Rete è generalmente più rapido rispetto ad un algoritmo naïve che prevede di ripetere ogni volta la query SPARQL e confrontare i risultati nel caso siano presenti un numero importante di sottoscrizioni (es. > 100). Impiega un tempo generalmente superiore ma circa uguale in caso contrario.

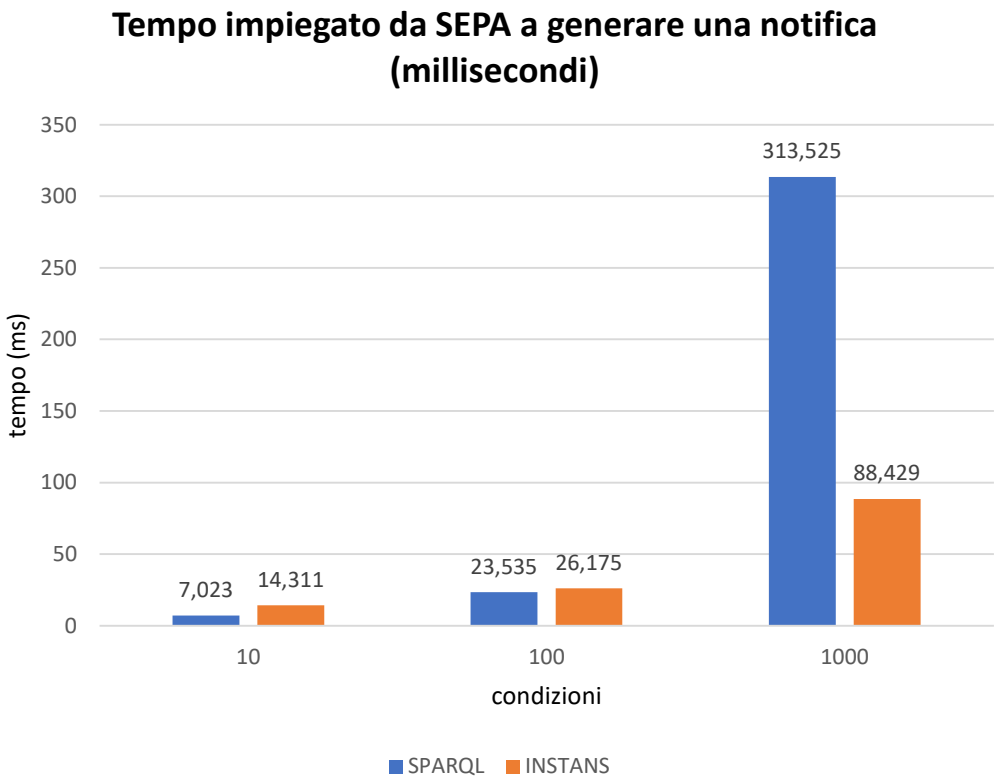

Grafico 3 Comparazione tempo di esecuzione SPARQL e INSTANS

3. CONCLUSIONI

L'algoritmo implementato garantisce un tempo di esecuzione inferiore rispetto al precedente nel caso in cui debbano essere effettuate query di grandi dimensioni, cioè nel caso di iscrizioni con un numero di pattern di triple elevato, rendendo molto più rapido l'inoltro delle notifiche degli aggiornamenti, richiedendo però una grande quantità di memoria. In caso contrario, il tempo impiegato a generare query di dimensioni ridotte, risulta superiore ma circa uguale rispetto a SPARQL.

Il codice prodotto per questa tesi è stato realizzato in linguaggio di programmazione ad oggetti Java, versione 11. Sono state utilizzate solamente librerie base, quindi è eseguibile su ogni macchina Windows, Mac, Linux, che abbia installato Java versione 11 o superiore. Il codice è pubblico e visualizzabile su GitHub al collegamento riportato alla sezione "Bibliografia e Sitografia" assieme alla documentazione HTML [13] [14].

Un possibile sviluppo futuro può essere:

- Effettuare un'operazione di "tokenization", ovvero salvare nella memoria dei nodi non l'intero elemento come avviene attualmente, ma un token associato ad esso, in modo che occupi meno spazio in memoria. Richiede una operazione di "de-tokenization" per ripristinare l'elemento originale nel momento deve essere portato in uscita.

BIBLIOGRAFIA E SITOGRAFIA

[1] "Internet delle cose," [Online]. Available: https://it.wikipedia.org/wiki/Internet_delle_cose.

[2] "Database," [Online]. Available: https://en.wikipedia.org/wiki/Database.

[3] "Query language," [Online]. Available: https://en.wikipedia.org/wiki/Query_language.

[4] W3C SPARQL Working Group, "W3C," 21 marzo 2013. [Online]. Available: https://www.w3.org/TR/sparql11-overview.

[5] "Web Semantico," [Online]. Available: https://it.wikipedia.org/wiki/Web_semantico.

[6] "Resource Description Framework," [Online]. Available: https://it.wikipedia.org/wiki/Resource_Description_Framework.

[7] "Uniform Resource Identifier," [Online]. Available: https://it.wikipedia.org/wiki/Uniform_Resource_Identifier.

[8] "Semantic triple," [Online]. Available: https://en.wikipedia.org/wiki/Semantic_triple.

[9] P. A. C. A. F. V. F. A. T. S. C. Luca Roffia, "Dynamic Linked Data: A SPARQL Event Processing Architecture," [Online]. Available: https://www.mdpi.com/1999-5903/10/4/36/htm.

[10] E. N. S. T. Mikko Rinne, "INSTANS: High-Performance Event Processing with Standard RDF and SPARQL," *CEUR Workshop Proceedings,* no. 914, pp. 101-104, Novembre 2012.

[11] "Rule-based system," [Online]. Available: https://en.wikipedia.org/wiki/Rule-based_system.

[12] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," 1982.

[13] M. Fontana, "Repository GitHub Rete," [Online]. Available: https://github.com/MarcoFontana48/rete-algorithm.

[14] "Repository GitHub SEPA," [Online]. Available: https://github.com/arces-wot/SEPA.

[15] "Uniform Resource Locator," [Online]. Available: https://it.wikipedia.org/wiki/Uniform_Resource_Locator.

RINGRAZIAMENTI

A mia mamma, per il sostegno e la motivazione che mi hai dato durante gli studi.

Al professore ingegner Luca Roffia e agli ingegneri Gregorio Monari, Gianluca Di Tuccio, rispettivamente relatore e correlatori di questa tesi, per la disponibilità e il supporto durante la progettazione e stesura.