Reliable SPARQL queries with consistent results over P2P-shared RDF storage

Pierluigi Di Nunzio, Federico Di Gregorio, Giuseppe Rizzo, Antonio Servetti
Dipartimento di Automatica ed Informatica
Politecnico di Torino
c.so Duca Degli Abruzzi, 24
10129, Torino, Italy
pierluigi.dinunzio@polito.it, federico.digregorio@polito.it, giuseppe.rizzo@polito.it, antonio.servetti@polito.it

Abstract: *One aim of the semantic web is to build large knowledge bases distributed over the internet. Knowledge management systems that gather, merge and make available the information physically stored in multiple locations suffer from consistency and data fragmentation issues due to node failures.*
In this paper we address such problems and we present an architecture for managing reliable SPARQL queries with consistent results over a P2P-shared RDF storage.
The RDF-storage is composed of peer nodes organized in a ring topology based on a Distributed Hash Table (DHT) where each node provides an entry point that enables clients outside the network to query the knowledge base using atomic, disjunctive, and conjunctive SPARQL queries. The consistency of the results is increased using a data redundancy algorithm that replicates each RDF triple in multiple nodes so that, in the case of peer failure, other peers can retrieve the data needed to solve the queries.
Additionally a load distribution algorithm is used to maintain a uniform distribution of the data among the participating peers by dynamically changing the key space assigned to each node in the DHT. The performance of this approach is then evaluated by monitoring the effectiveness of the load balancing and redundancy algorithm and the overhead introduced on the network load in both a static (only join events) and a dynamic scenario.

Keywords: RDF, storage RDF, RAID RDF, Peer-to-Peer RDF, semantic Peer-to-Peer storage

1. Introduction

A key focus of Semantic Web is the possibility to create new knowledge by sharing semantic information originated by multiple sources using a common framework based on standard data formats and network protocols. In such a scenario information is frequently distributed on a very large number of nodes over the whole Internet.
When the information is distributed, multiple peers are involved in a single query and results may change over time, i.e., if a network node is not available, the retrieved data set will not include the related information. Also, if the information is randomly distributed over the whole network the query response time will increase proportionally with the number of nodes. In interactive services, when knowledge is meant to be used by a human being, these problems are critical because the user will expect coherent results in a reasonable response time. Our work will address these data consistency and availability issues introducing redundancy and load balancing algorithms.
In our proposal, nodes are self-organized in a P2P network and, in addition to their own data; they also store a redundant subset of the information originated by the other peers. Taking as a basic unit of information the RDF triple that represents a statement where a predicate denotes a relationship between a subject and an object, the problem is to find a distribution algorithm to efficiently store multiple copies of each triple and to retrieve it in predictable time independently of the temporary unavailability of the original source. The main focus of this paper is to discuss a solution that involves a triple distribution algorithm, a triple validity management policy and a peer-to-peer (P2P) routing infrastructure.
This paper is organized as follows. In the next Section, we analyze the state of art in this research area. In Section 3 we describe the proposed architecture and the major issues related to node insertion/removal and information redundancy. In Section 4 and 5 we discuss the results and present our conclusions and future works.

2. Related Work

Many efforts have been made to build an infrastructure composed by several centralized peers aimed at sharing knowledge and support consistency and availability of retrieved data. Jena [6] and Sesame [7] are examples of centralized storage systems which enable external applications to retrieve data using the SPARQL protocol [1], a graph-matching query language used to retrieve semantic knowledge. A set of this storage servers can cooperate together to share knowledge, but the major issue related to this approach is that the results may change over time, as a function on the availability of network nodes. That is, if a node becomes unavailable it is not backed up by any other node of the network. Thus any node may be a single point of failure for the system [19].
To overcome these centralized constraints, many approaches in literature consider a distributed scenario, based on a P2P network. A P2P approach is proposed in Edutella [14] using the Gnutella protocol. The overlay is composed of a set of Super Peers (SP) [15], each of them connected to a large number of simple peers. The SPs are nodes with high network bandwidth and sustainable computation power. These SPs form the backbone, manage a local set of RDF data and are responsible for routing and querying. Although this approach is developed for a distributed scenario, it also

presents the single point of failure issue for the SPs, because when a SP is not available, the consistency of the retrieved knowledge is compromised.
Another approach that considers a distributed scenario is RDFPeers [4], a spanning of RDF-Repositories over a Peer-to-Peer network, based on a DHT, which is developed on top of Multi Attribute Addressable Network (MAAN) [5], an application layer based on Chord [21]. Each peer in this network is responsible for a segment of the whole key space, thus satisfying the “node equality” principle in terms of required network bandwidth, storage and query dependent computational load. Each RDF field is hashed and a copy of the triple is stored in the nodes responsible for the segments where the hashes fall. Any query can then be resolved by hashing one of the constraints and contacting the node responsible for that hash ID. This approach provides load balancing based on the hash function, which assigns an ID to each input data, but it does not enforce the uniformity of such resulting values, so we may end up having a very loaded sector in the network, while others are lightly loaded. On the long run, in fact, the non-uniform distribution of the semantic statements, e.g. the high frequency of rdf:type or dc:title statements, will concentrate a lot of data in the particular sectors matching those hash ID. To overcome this issue, RDFPeers uses a successor probing algorithm which randomly analyzes a sub-set of all ranges and chooses the heaviest to be divided. RDFPeers interprets the atomic, disjunctive and a sub-set of conjunctive queries but does not support SPARQL as query language. In addition no active redundancy system ensures the consistency of the retrieved RDF data.

3. System Architecture

The typical scenario considered in this article is a network composed of hundred peers each with its own RDF store. Each peer is both a “server”, because it shares its knowledge base with the network, and a “client”, because it makes use of the knowledge base shared by the other peers. In a distributed knowledge base (built sharing the resources of many single nodes), this implementation can increase its capacity proportionally with the number of nodes in the network, differently from centralized systems.
More precisely, each peer is authoritative for the information stored into its own RDF database and, at the same time, it provides a backup storage space for RDF triples belonging to the other peers of the network. In fact, as a function of a specific redundancy algorithm, each node replicates its RDF triples into other nodes of the network in order to guarantee an increased degree of reliability of the system to disruptions.
Furthermore, at any time any server may decide to join or leave the network, thus adding or marking as invalid some of the distributed information. Although the accessibility of a given node in the network cannot be guaranteed at all times, to satisfy the data consistency and availability requirements, the information owned by the node should be available at least within a specific time interval (RDF triple time-to-live). This scenario suggests the use of a structured P2P network where the peers self-organize in a simple topology, so that network changes due to frequent node join and leave events can be performed with little overhead. In fact, more complex topologies or hierarchical models, that guarantee faster routing and search operations, require too much work for network reorganization and are not suitable for such a dynamic environments.
RDF triples are indexed by multiple hash values calculated on the subject, predicate and object. Literal and typed literal values are excluded and each triple can yield from one to three hash values. Hash values are then used to locate a bucket node in a single ring according to the Mercury protocol [2]. We assign to each node a segment of the key space which can grow or reduce in order to provide uniform triple distribution, even if data is concentrated in a short key range, so every node in the network is responsible for nearly the same amount of data. Using periodical routing messages, peers exchange information on the local triple distribution in order to monitor the range with the highest load. Then, when a new node joins the network, the system uses that node to split the most loaded range so that the two nodes become authoritative for half of the triples previously contained.

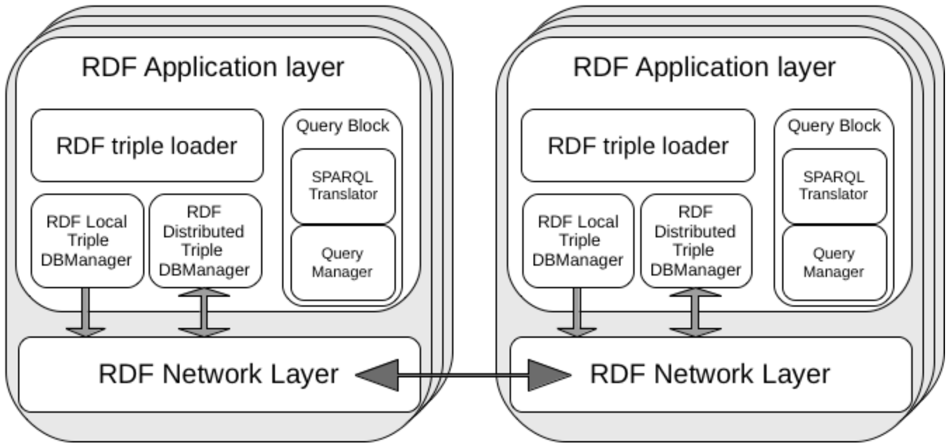

Figure 1: System Architecture.

As can be seen in Fig 1, each server is organized in a stack composed by a network layer and an application layer. The underlying network layer manages the routing information in the P2P network and also manages the join/leave events and the load balancing. The application layer is composed of the RDF triple loader, used to read the RDF documents and store the information into the local RDF triple database, a distributed RDF triple database used to store information shared with the other nodes and a query block used to interpret the SPARQL queries.

In the next subsections we focus on the RDF network layer. We explain the model used for message routing, the algorithms used to estimate the node load and the techniques implemented to manage the node join and leave events.

3.1 Message routing

Following the idea proposed in [2], and shown in Fig. 2, each node has a DHT composed of two sibling links connected to the previous and next node in a clockwise direction and k long distance links chosen so that the distance from the source node to the target node follows a harmonic law. This scheme provides good efficiency so that we route a generic message between two nodes in $O(\log^2(n)/k)$ [13], where k is the number of long distance links [12], that is $\log(n)$ as shown in [2]. In Fig. 2 we analyze a six bit network, where each node has a DHT composed of six long distance links. In addition, we can observe how a generic routing message is delivered between the nodes.

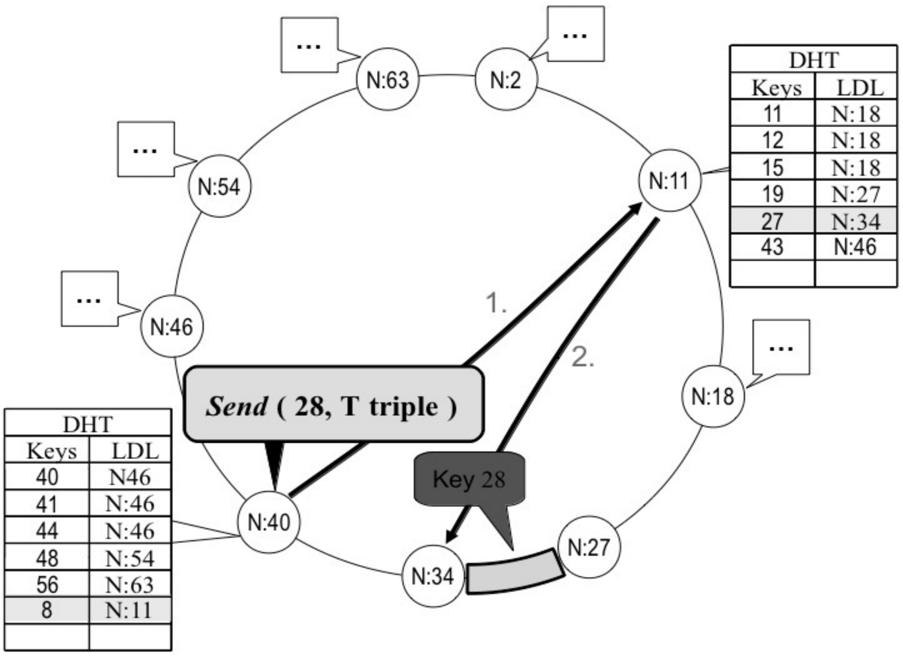

Figure 2: Routing one triple into our network with nine nodes in a 6-bit identifier space.

A generic node m builds k long distance links with the following procedure; let I denote the unit interval $[0,1]$ (that corresponds to the DHT ring with unit perimeter), for each such link the node draws a number $x \in I$ from the harmonic probability distribution function $p_n(x) = 1/(n \times \log(x))$, if $x \in \left[\frac{1}{n},1\right]$, where n is the number of nodes in the network. Then it establishes a long distance link with a node that is distant $p_n(x)$ from itself on the DHT ring.
As shown in Fig. 3, if it happens that the procedure uses inconsistent DHT information, the message routing algorithm can incur in routing loops as described in [8].
This can be avoided having each hop i checking the ID of the node from which it has just received the message to be forwarded (the source) so that, if the ID value of the target node is between the ID of the source node and the ID of the previous (i) node, then it forwards the packet via the backward link. In this case the routing cost is bounded to $O(n)$.

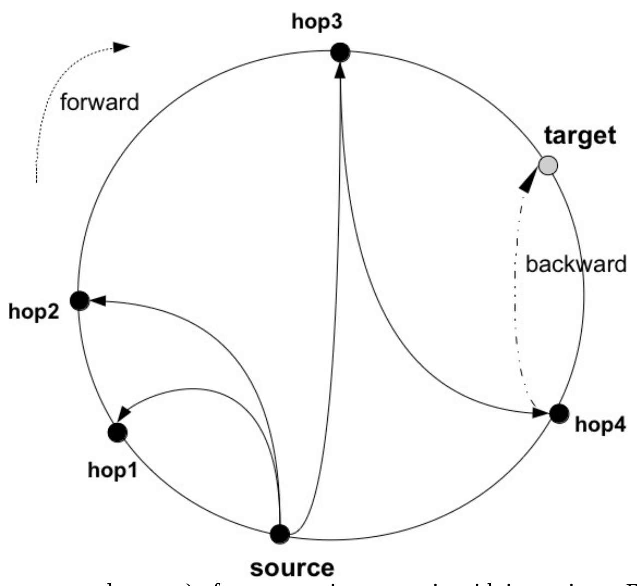

Figure 3: A backward-routing (*source* towards *target*) of a message in a scenario with inconsistent DHT information.

3.2 Consensus Based distributed network load estimation

In order to maintain a uniform distribution of the data among the peers, some of the algorithms proposed in this work rely on the estimation of the load of the network nodes. For this purpose we use a distributed algorithm that relies only on the communication between neighbouring nodes in the network graph and that adopts a consensus algorithm for the election of the most loaded node.
It has been shown in the literature that the convergence speed of consensus problems is a function of the topology of the underlying graph. While this bound can be easily calculated for fixed networks, this is not the case for dynamic topologies such as a time-varying P2P network. However, it has also been shown that, as long as the union of all infinitely occurring graph instances is connected, there is a distributed consensus that will eventually converge [24] and that good convergence speed can be achieved imposing particular constraints on the different topologies.
As described in Sec. 3.1, the considered P2P architecture is composed of a cloud of nodes with a regular topology where distant peers are connected by means of a number of additional short paths. Such “long distance links” allow the characterization of the network topology as a “small world”, a concept firstly introduced by Watts and Strogatz [23] a phenomenon observed in many real-world graphs such as large-scale social, biological and technological networks [22]. A reasonable conjecture is that the small world graphs should result in good convergence speed for consensus problems because their low average pair wise path length should speed the diffusion of information in the system [10]. Based on the small world principle we consider a typical consensus problem to determine a convergence criterion for the election of the most loaded node. Consider a network of integrator agents $\dot{x}_i(t) = n_i$, where n_i is a generic node within the network and let $G = (V, E)$ be a graph with the set of nodes $V = v_1, \dots, v_n$, the set of edges, $E = V \times V$ and in which each node only communicates with its neighbouring nodes $N_i = \{ j \in V : \{i, j\} \in E \}$.
Following the idea proposed in [17] and [18] the linear dynamic system:

$$\dot{x}_i(t) = \sum_{j \in N_i} (x_j(t-\tau) - x_i(t-\tau)), \qquad (1)$$

solves a consensus problem, where τ is the time-delay for all links that we suppose, for simplicity, constant.
More precisely, let $a_1, \dots, a_n \in \Re$ be n constants, then with the set of initial states $x_i(0) = a_i$, the state of all agents asymptotically converges to the average value:

$$\bar{a} = \frac{1}{n} \times \sum_{i=1}^{n} a_i \ . \qquad (2)$$

A necessary and sufficient condition for the stability of Eq. 1 is given in [17] as:

$$\tau < \tau_{\max} = \frac{\pi}{2 \times \lambda_n}, \quad (3)$$

where λ_n is a measure of robustness to delay for reaching a consensus in a network. As a consequence, to guarantee the convergence of the algorithm for the selection of the most loaded node, we define a bound on the maximum delay of the load estimation algorithm limiting the recursive search on neighbour nodes to a depth given by a TTL value. Thus, we extend the result in Eq. 1 as:

$$\dot{x}_i(t) = \sum_{j \in N_i} (x_j(t - \tau - TTL) - x_i(t - \tau - TTL)). \quad (4)$$

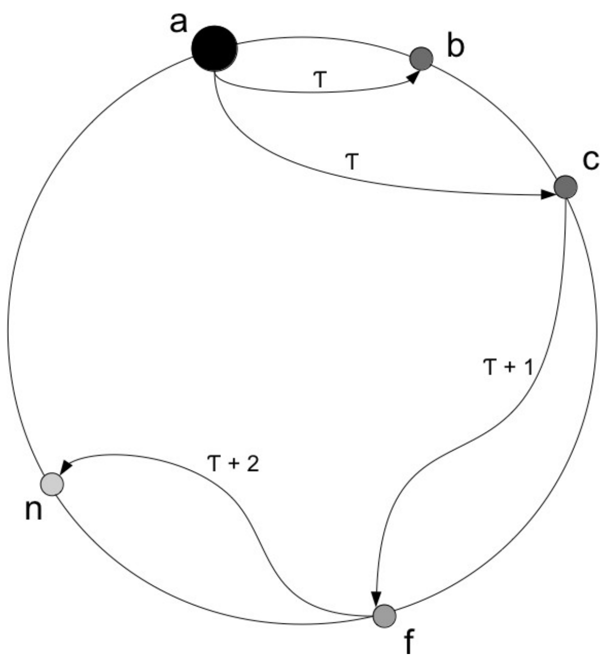

Figure 4: Each node contacts our directly neighbours (nodes which are indexed from the DHT) in τ seconds. Let's n the distant factor, a generic node contacts a not directly neighbours in a $n \times \tau$ seconds.

3.2.1 Load estimation procedure

As shown in Fig. 4 in order to compute a realistic load histogram of the network, each node periodically contacts its neighbours, listed in the DHT table, using "load estimate" packets.
The packet format is shown in Fig. 5 and is composed of: (1) Time-to-Live (TTL), which defines the number of hops to discover and it represents the length of the short path, (2) Timestamp, which identifies when estimate has been calculated, (3) consensus-range-begin and (4) consensus-range-end, which describe the leftmost and rightmost key of sample estimate of the projection load estimate of the current node, (5) number of nodes and (6) average-load, which identifies the total number of triples divided by the number of total nodes, and (7) most-load-node, which describes the most loaded node in the macro-range. In order to reduce the bandwidth overhead we propose to append the "load estimate" information to the routing messages used to manage the network.

TTL	Timestamp	consensus range begin	consensus range end	number of nodes	average load	most loaded node

Figure 5: Load estimation packet format.

Nodes recursively propagate these packets using their long distance links and the TTL field is decremented by 1 for every hop. When the TTL value becomes equal to zero, the packet is sent backward and it returns to the original

sender, i.e. the source node (node *a* in Fig. 5). In this way, each node can receive a feedback on the load estimation done by each of its neighbours with a delay that can guarantee the convergence of the distributed consensus algorithm.
To overcome the initial state problem, the consensus algorithm performs the network load capture periodically every τ seconds and then it only takes the estimate of the nodes which directly referenced to the source node. Furthermore, because of the dynamic nature of the network, old load estimates risk to become counterproductive. Thus estimates older than a given threshold must be deleted or updated. In order to perform this, a *timestamp* reference is used: when this value becomes older than a threshold then a new load estimation procedure is performed.
As a result of the load estimation process nodes can calculate the distribution of the load among the peers and the number of peers in the network.

3.3 Join and leave events

When a new node A wants to join the P2P network, it contacts a known node B that, as a function of the estimated load information, redirects A to the most loaded peer it is aware of, C. The joining node A is then inserted in the ring as the predecessor of C. As a consequence, the data range previously managed only by C is evenly split between C and A, i.e., A acquires half of the triples that have been on C, according to the principle of consistent hashing [11]. Obviously, groups of triples stored under the same hash could not be split and therefore should be considered as grains among the triples. Since the triples are stored sorted by their numerical value, C flushes each data whose hash is greater than the ID assigned to the new node.
The leave event can require two different actions since we may want to preserve the consistency of the data or we may want to remove portion of the knowledge base from the network. In the former case the event requires that the leaving node transfers its triples to its successor (following the principle of consistent hashing) that has to update its responsibility range. In addition, the leaving node forwards through its long-distance links a leave message so that these peers can update their DHT to match the new state of the network. In the latter case, the leaving node may also want to remove from the network the triples it is owner of, so it requests the deletion of these triples and their redundant copies. In the rest of the paper we consider as a leave event only the case that can also be forced, for example, by a link failure or temporary node unavailability.

3.4 Redundancy

A redundancy algorithm is proposed to address the issues related to temporary node unavailability (or node disconnection). Triples are in fact replicated in more than one peer according to their hash function, using the following equation:

$$\sum_{i=0}^{t-1} hash(x) + \frac{w \times i}{t}, \tag{5}$$

where x is a URI field of a statement, ω is equal to 2^{hash_length} and t identifies the number of replicas. Fig. 6 illustrates an example of the redundancy mechanism, where each triple has an additional copy in another peer of the network which is chosen using Equation (5) with t = 2.
This algorithm does not group all triples replica of a node in one peer. When the network must substitute a node that has left, it uses the above formula to transfer the redundant triples of the dead node to its successor. In this case the node that receives the triples does not need to replicate them again.

3.5 Query resolving

In the proposed system any client outside the P2P network can use the SPARQL standard, a graph-matching query language used to retrieve semantic knowledge, on any P2P node to query the network knowledge base. The distributed RDF architecture can then return the same results in a predictable time independently from the particular node used to perform the query.
The basic idea that we use for querying the knowledge base is to hash the exact value of at least one triple field. Then, these hashes identify the nodes which store the requested triple-set. In the case of multiple exact fields available in a query, we use a query parsing algorithm for rewriting the query to minimize the cost of answering it. This cost is affected by at least three elements: the size of the knowledge base (in particular the number of nodes which are involved in the query), the strategy followed to combine the data and the order or plan in which data is processed. In our context, i.e., a highly dynamic P2P scenario, new techniques to identify an efficient query evaluation strategy are needed.

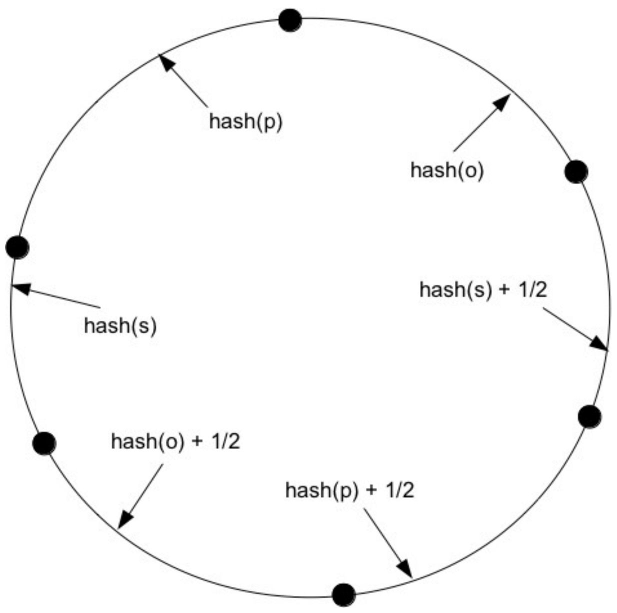

Figure 6: Redundant copies of a given triple (s, p, o) with $t = 2$.

For what concerns the query parsing algorithm, the SPARQL Query Graph Model (SQGM) [9] has been proposed as a method to build an optimizer such that it can elaborate a right strategy for combining the middle results. However, this approach relies on a central node for building a global graph and it requires the collection and transfer of a huge amount of data across the P2P network. On the contrary, we suggest using a distributed algorithm since we have a global knowledge base composed of a pool of P2P nodes which make unfeasible to share a global graph as in the SQGM proposal.
In [20] a most efficient technique has been presented that constructs optimized heuristics as a function of the number of occurrences of an element within the knowledge base, i.e., as a function of the element *selectivity*. Although this implementation has been developed for a centralized environment, we extend it for a distributed scenario where the network is able to collect the selectivity information of all elements owned by the P2P nodes.
To evaluate the performance of query execution, we consider the query propagation time between nodes. We assume an upper-bound delay for the propagation time between two nodes, which is called τ , and that, for simplicity, we suppose constant and equal for all nodes. Thus, we express the total query time as:

$$QT = \sum_{i=1}^{n} C_i + R , \qquad (6)$$

where C_i is equal to the sum of the computing time of all nodes and R is equal to the routing time.
In the rest of the section we analyze three types of queries: atomic, disjunctive and conjunctive queries.

3.5.1 Atomic query

An atomic query pattern is a triple in which the subject, predicate and object can each be a variable or an exact value. In this case, the query block just uses the hash function on the query pattern exact values to identify the node which manages the requested triple-set. An example of an atomic query is represented by the form (?s, p, o), where the question mark indicates the unbound or variable value. Following the idea proposed in [20] we define the selectivity of a triple pattern as $s(t) = s(s) \times s(p) \times s(o)$, where $s(t)$ denotes the selectivity for the triple pattern t , $s(s)$ the selectivity for the subject s , $s(p)$ the selectivity for the predicate p , and $s(o)$ the selectivity for the object o . The (estimated) selectivity is a real value in the interval $[0,1]$ and corresponds to the (estimated) number of triples matching a pattern, i.e., the number of triples matched, normalized by the total number of triples in the RDF data-set. The selectivity of a unbound triple field, i.e., s(s) or s(p) or s(o), is generally equal to 1.0 , otherwise the selectivity of an exact value is equal to T_e / T , where T_e denotes the number of occurrences of a triple element and T the total number of triples in the entire knowledge base. Thus, a low selectivity value means a high element selectivity and vice versa.
A general representation of an ontology, which use the RDF notation for representing its data, is generally composed of an absolute number of subjects at least one order of magnitude greater than the absolute number of predicates [16].

According to Eq. (6), $\sum_{i=1}^{n} C_i = s(e) \times \{C_s + C_t\}$, where $s(e)$ is the selectivity of the element and C_s and C_t are, respectively, the computing time of the source node and of the target node. In addition, since R depends linearly on the number of nodes interested in the query times τ and the routing protocol guarantees that a generic node can be reached in $a = O(\log^2(n)/k)$, we can state that R is upper-bounded by $a \times \tau$. As a consequence, the computation time of the target node depends to a great degree on the number of triples owned by a node and on the number of potential selectable triples. For this reason, to minimize the computation time required, the query is sent to the node with the higher selectivity
A different case is represented by the queries in the form (?s, ?p, ?o). This is the most general and the most expensive query which matches all triples. Since there is no restriction whatsoever on this triple pattern, we have to contact all nodes, which takes $O(n)$ routing hops for a network with n nodes. For this reason, the $QT = C_s + n \times \tau$, where C_s is the source node computing time.

3.5.2 Disjunctive query

A disjunctive query pattern is an atomic query with a list of constraints. It is then evaluated almost as an atomic query, but, here, the node responsible for such triple-set has to return only the data that matches the specified constraints. For example, consider these two queries:

(a) (?s, foaf:name, ?o),
?o = ``Picasso" OR
?o = ``Modigliani"
(b) (?s, media:length, ?o),
?o > ``54" AND
?o < ``99"

They are divided into two separate atomic triple patterns and they are sent to two different nodes. As described previously the single queries are sent to the nodes with highest selectivity, the results are then collected by the source node.
Let N the number of atomic queries generated, following Eq. (6) we have,

$$QT = \sum_{i=1}^{n} C_i + \tau \times d \ , \quad (7)$$

where d is upper-bounded by $O\left(N + \log^2(n)/k\right)$ and C_i is the value of the computation time of the i -th node.

3.5.3 Conjunctive query

A conjunctive query pattern, or simply a joint query, is the conjunction of a list of query patterns. Consequently, we need to join two or more sets of triples that are spread on different nodes. We adopt the query chain algorithm (QC), described in [7]. For example, consider this query:

(?x, foaf:name, "Johnny Lee Outlaw"),
(?x, foaf:mbox, ?mbox)

This schema splits the whole query in sub-queries and solves them separately as an atomic query. The result of the sub-queries are then collected and joined to form the requested triple-set.
Assume a node j that poses a conjunctive query q which consists of triple patterns $q_1, \ldots, q_k$. Each triple pattern of q will be evaluated by a (possibly) different node; these nodes form the query chain for q . The order we use to evaluate the different triple patterns is decided by the query block, which rewrites the query in order to optimize the number of triples selected by the first nodes. Thus, a source node issues the most constraint query to the first node. Then, this one issues the rest of the query and the result of the first sub-query to second node. The last node in this chain computes the triples required and sends the results to the source node. In this fashion we reduce the number of triples exchanged between the source node and the nodes involved into the query.
Let N be the number of sub-queries, from Eq. (6) we have:

$$QT=\sum_{i=1}^{n} C_i+\tau\times c \ , \tag{8}$$

where c is upper-bounded by $N\times O\left(\frac{\log^2(n)}{k}\right)$.

4. Experimental results

To asses the performance of the proposed architecture we developed a specific simulator that reproduces the behaviour of a medium size network with three hundred nodes and one million of triples in the data-set.
First, we focus on the analysis of the load balancing technique; we study the distribution of the triples among the peers and then the network bandwidth overhead introduced by this feature. Two basic scenarios are considered, a) static, where nodes sequentially join the network monotonically increasing its size, b) dynamic, where nodes join and leave the network. In each join event the new node inserts 3,334 new triples, which is the average value of triples in each node of the existing network.

4.1 Load balancing

In Fig. 7 and Fig. 8 we represent the relative standard deviation (RSD) of the number of triples managed by each peer in two simulation scenarios.
In Fig. 7, 300 nodes are sequentially inserted in the P2P network and the RSD value is reported after each insertion. When a node joins the network, 3,334 triples are added to the system and the new node becomes the predecessor of the most loaded one inheriting half of its triples. The plot shows that, apart from the very beginning of the simulation when few nodes are present, the relative standard deviation is kept almost constant ensuring a good load balancing among the peers. Variations are mostly due to the uneven distribution on the P2P network of the new 3,334 triples that come with the joining nodes.
In Fig. 8, after the insertion of 300 nodes, the graph represents the effect on the RSD of ten series of ten join and ten leave events. Peaks correspond to the leave events when the leaving node triples must be moved to the successor node drastically increasing its load with respect to the other peers. Valleys are instead related to the join events when a new peer joins the network and helps splitting the triple range managed by the most loaded peer. Since each joining node carries a relative large amount of new random triples the RSD delta after a series can either be positive or negative depending on the new triple hashes. For example series I, II, III, IV (from event 300 to event 380) increase the RSD, while series V decreases it. However series that increase the RSD tends to occur more frequently, because, as previously shown in Fig. 8, the bandwidth saving load balancing algorithm applied to the insertion events can only slowly reduce the RSD.

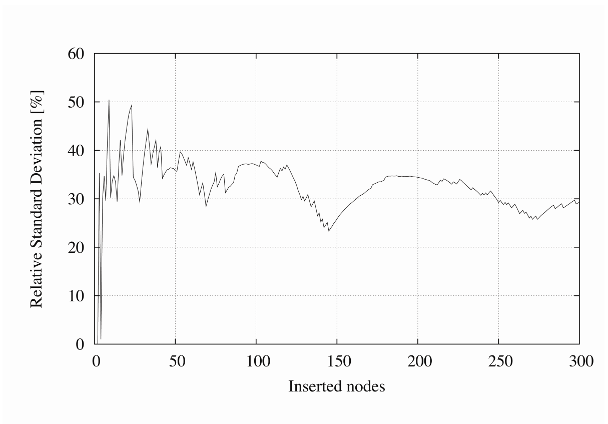

Figure 7: Load balancing analysis with only join events. Few insertions tend to make the P2P unbalanced, while many insertions optimize the uniform triple distribution.

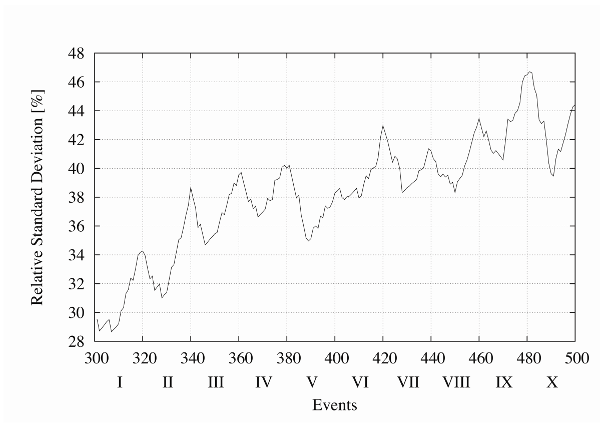

Figure 8: Load balancing analysis with 10 series of join and leave events. Valleys are due to leave events, because of all node triples are moved to its successor, while peaks are due to join events, because of each node distributes its owned triples and gets half of the MLN node.

4.2 Network bandwidth

For what concerns the network bandwidth used by the proposed algorithm to guarantee load balancing and triple redundancy, the effects of data exchange in the two previous scenarios (static and dynamic) are show in Fig. 9 and 10.
The number of triples exchanged is normalized with respect to the number of triples each node inserts in the system (e.g. 3,334). So, when a new node joins the network most of its triples move into the ranges managed by other peers and half of the triples of the most loaded peer moves into the node hash range. When a node leaves the network its triples move to the node successor.

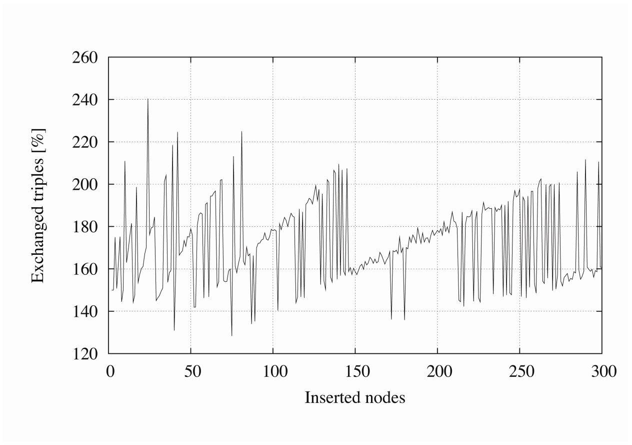

Figure 9: Network bandwidth analysis in a static scenario. Peaks are due to join of new nodes, because of a node splits the MLN range and the distribution triples.

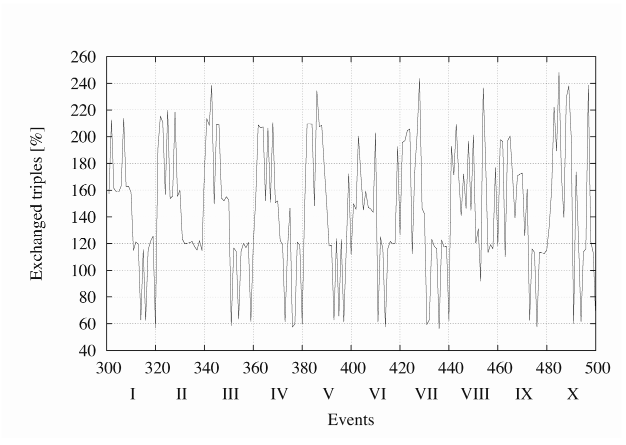

Figure 10 : Network bandwidth analysis in a dynamic scenario. Peaks are due to join event, because of each node gets half of MLN triples and send its triples, while valleys are due to leave event, because of node moves its triples to the successor.

5. Conclusions and future works

In this paper we proposed a distributed RDF triple storage based on a P2P network. Triple distribution among the network peers is optimized in order to have each node responsible for nearly the same amount of data. Each field of the triples, excluded the literal, is hashed and distributed according to the long distance links that are defined using the estimated network load information. In addition we address the problem of network failures and frequent node join and leave events with a redundancy mechanism, i.e. a portion of each peer storage is used to store copies of triples managed by the other peers. Thus, we ensure, to a certain degree, that if a link fails or a node abruptly disconnects from the network, no triples are lost and a query is able to return consistent results. The performance of this approach is measured by monitoring the effectiveness of the load balancing algorithm and the overhead introduced on the network load in both a static (only join events) and dynamic scenario.
Future work will further investigate the problem using a large training-set of real data, like a complete set of Wikipedia triples, and a high dynamic scenario with heterogeneous peers and random leave/join events.
Besides we will also investigate how a highly dynamic P2P network, where peers frequently join and leave, impacts the efficiency of the RAID-like system we developed. First, we will try to determine the optimal ratio between the number of nodes and the total number of triples distributed over the network: the performance of joint queries depends on the balance between the number of nodes to be contacted and the amount of data managed by each node. Finally we will analyse the main issues related to query failures like too many unreachable nodes and TTL expiration, and how to mitigate them.

Acknowledgements

We wish to thank Andrea Benazzo for his initial studies on building the RDF infrastructure upon a set of P2P protocols.

References

[1] Prud'hommeaux, E., Seaborne, A. (2008). SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, January 2008.

[2] Bharambe, A. R., Agrawal, M., Sehan, S. (2004). Mercury: supporting scalable multi-attribute range queries. SIGCOM Comput, Commun, Rev., 34(4):353-366, 2004.

[3] Broekstra, J., Kampman, A., Van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying RDF and RDF schema. In International Semantic Web Conference, pages 54-68, 2002.

[4] Cai, M., Frank, M. (2004). RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In Proc. of the 13th international conference on World Wide Web pages 650-657, New York, NY, USA, 2004, ACM.

[5] Cai, M., Frank M., Chen, J., Szekely, P. (2003). MAAN: A multi-attribute addressable network for grid information services. IEEE/ACM International Workshop on Grid. Computing, 0:184, 2003.

[6] Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K. (2004). Jena: implementing the semantic web recommendation, In Proc. of the 13th international World Wide Web conference on Alternate track papers and posters, pages 74-83, New York, NY, USA, 2004, ACM.

[7] Liarou, E., Idreos, S. and Koubarakis, M. (2006). Evaluating conjunctive triple pattern queries over large structured overlay networks. In Proc. of the International Semantic Web Conference, pages 399-413, 2006.

[8] Freedman, M. J., Lakshminarayanan, K., Rhea, S., Stoica, I. (2005). Non-transitive connectivity and DHTs. In Proc. of the 2nd Conference on Real, Large Distributed Systems, pages 55-60, Berkeley, CA, USA, 2005. USENIX Association.

[9] Hartig, O., Heese, R. (2007). The SPARQL query graph model for query optimization. In Proc. of the 4nd European conference on The Semantic Web, pages 564-578, Berlin, Heidelberg, 2007. Springer-Verlag.

[10] Hovareshti, P., Baras, J. (2006). Consensus problem on small world graphs: a structural study. In Proc. of the International Conference on Complex Systems, 2006.

[11] Karger, D. , Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D. (1997). Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In Proc. of the twenty-ninth annual ACM symposium on Theory of computing, pages 654-663, New York, NY, USA, 1997. ACM.

[12] Kleinberg, J. (2000). The small-world phenomenon: an algorithm perspective. In Proc. of the thirty-second annual ACM symposium on Theory of computing, pages 163-170, New York, NY, USA, 2000. ACM.

[13] Manku, G. S., Bawa, M., Raghavan, P., Verity Inc. (2003). Symphony: Distributed hashing in a small world. In Proc. of the 4th USENIX Symposium on Internet Technologies and Systems, pages 127-140, 2003.

[14] Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T. (2002). EDUTELLA: a P2P networking infrastructure based on RDF. In Proc. of the 11th international conference on World Wide Web, pages 604-615, New York, NY, USA, 2002. ACM.

[15] Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., Loser, A. (2003). Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In Proc. of the 12th international conference on World Wide Web, pages 536-543, New York, NY, USA, 2003. ACM.

[16] Neumann, T., Weikum, G. (2009). Scalable join processing on very large RDF graphs. In Proc. of the 35th international conference on Management of data, pages 627-640, New York, NY, USA, 2009. ACM.

[17] Olfati-Saber, R., Murray, R. (2004). Consensus problems in networks of agents with switching topology and time-delays. IEEE Transactions on Automatic Control, 49(9):1520-1533, 2004.

[18] Saber, R., Murray, R. (2003). Consensus problems for networks of dynamic agents. In Proc. of the American Control Conference, volume 2, pages 951-956, 4-6, 2003.

[19] Steffen Staab, H. S. (2006). Semantic Web and Peer-to-Peer. Springer, 2006.

[20] Stocker, R., Seaborne, A., Bernestein, A., Kiefer, C., Reynolds, D. (2008). SPARQL basic graph pattern optimization using selectivity estimation. In Proc. of the 17th international conference on World Wide Web, pages 595-604, New York, NY, USA, 2008. ACM.

[21] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashock, M. F., Dabek, F., Balakrishan, H. (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw., 11(1):17-32, 2003.

[22] Tahbaz-Salehi, A., Jadbabaie, A. (2007). Small world phenomenon, rapidly mixing markov chains and average consensus algorithms. In Proc. of the 46th conference on IEEE Decision and Control, pages 276-281, 2007.

[23] Watts, D. J., Strogatz, S. H. (1998). Collective dynamics of small-world networks. Nature, 393(6684):409.-10, 1998.

[24] Xiao, L., Boyd, S., Lall, S. (2006). A space diffusion scheme for peer-to-peer least-squares estimation. In Proc. of the 5th international conference Information processing in sensor networks, pages 168-176, New York, NY, USA, 2006. ACM.

Pierluigi Di Nunzio 36 years old, Italian, free software consultant since 1999. Experienced in web applications development and usability, system integration and database engineering. International expert on government portals. His research focuses on semantic web authoring tools.

Federico Di Gregorio 37 years old, Italian, free software consultant since 1999. Experienced in network and web applications development, system integration, security and database driver programming. International expert on government electronic networks. Active contributor toward many FOSS projects and developer for the Debian project since 1997.

Giuseppe Rizzo received his master degree in Computer Science Engineering from the Politecnico di Torino, Italy, in 2008. His thesis topic was about clustering and automatic classification of electronic mails, advised by professors A.R. Meo, F. Di Gregorio and P. Di Nunzio. He's currently working on semantic web area, with the specific focuses in P2P network and RAID solution for storing RDF tuples. His interests also comprise internet applications, machine learning and artificial intelligence implementations.

Antonio Servetti received the Ph.D. degree in computer engineering from the Politecnico di Torino, Italy, in 2004. In 2000 he was with CSELT (now Telecom Italia Lab) in Turin, then in 2003 he was a Visiting Researcher at the Signal Compression Laboratory, University of California, Santa Barbara, where he worked on speech compression algorithms. He is currently an Assistant Professor at the Department of Control and Computer Engineering of the Politecnico di Torino. His primary research interests include speech and audio processing, multimedia coding and transmission over wireline/wireless IP networks. Dr. Servetti is co-author of over 20 international scientific publications and co-inventor of a US patent on fast multimedia audio decoding.