A Peer-to-Peer Architecture For Distributed And Reliable RDF Storage

A Peer-to-Peer Architecture For Distributed And Reliable RDF Storage

Giuseppe Rizzo, Federico Di Gregorio, Pierluigi Di Nunzio,
Antonio Servetti, Juan Carlos De Martin

Dipartimento di Automatica e Informatica
Politecnico di Torino, Italy
E-mail: `<name.surname>@polito.it`

Abstract

Knowledge management systems share information from multiple sources over the network and may have problems in maintaining the consistency due to node failures and data fragmentation in different locations. In this paper we present an architecture for a load balanced and reliable RDF storage system for semantic information distributed over a peer-to-peer network.

Peers are self organized in a ring topology, based on a Distributed Hash Table (DHT), where each node is assigned a segment of the key space that can dynamically change in order to maintain a uniform distribution of the data among the participating peers. Data redundancy is then used to replicate each RDF triple in multiple locations so that, in case of peer failures, neighbour nodes can act on their behalf and return consistent results. Additionally, each node provides an entry point able to resolve atomic, disjunctive and conjunctive SPARQL queries on the network semantic knowledge.

The performance of this approach is 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 dynamic scenario.

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 a RAID-like and load balancing policy.

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 [3] 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 [15].

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 [13] using the Gnutella protocol. The overlay is composed of a set of Super Peers (SP) [14], 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], based on a DHT, which is developed on top of Multi Attribute Addressable Network (MAAN) [5], an application layer based on Chord [16]. 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. 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 several hundred peers each with its own RDF store. Each peer is authoritative for the information stored into its own RDF database and, according to a specific distribution algorithm, it sends multiple copies of each RDF triple to other nodes in the network. At the same time each server provides storage space for RDF triples originated by other peers in the network. 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 can’t 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’s time-to-live). This scenario suggests the use of a structured P2P network where the peers self-organize in a ring topology and use a DHT to build a distributed and highly reliable RDF triple repository.

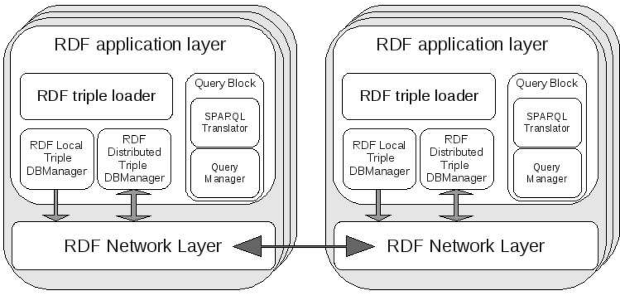

Figure 1. System architecture.

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. 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 algorithms used to estimate the node load, to manage the node join and leave events and the

TTL	TimeStamp	macro range start	macro range end	average range	average load	most loaded node

Figure 2. Load estimate packet format.

model used for message routing.

3.1. Load estimation algorithm

In order to compute a realistic load histogram of the network, nodes periodically contact other nodes, which are chosen according to our DHT, using "load estimate" packets. The packet format is shown in Fig. 2 and is composed of: (1) TimeToLeave (TTL), (2) timestamp, which identifies when estimate has been calculated, (3) macro-range-start and (4) macro-range-stop, which describe the leftmost and rightmost key of macro-range, (5) average-range and (6) average-load, which identify the average range and the average load among the nodes and (7) most-load-node, which describes the most loaded node in the macro-range.

Once a node receives this packet, it appends its load estimate, decrements the TTL field and forwards it again according to the long-distance links, chosen from the DHT via a random uniform probability distribution. When the TTL value is equal to 0, the last node sends such estimates to the source node. In order to reduce the bandwidth overhead we propose to append the "load estimate" information to the routing messages used to manage the network. Because of the dynamic nature of the network old load estimates, there is a slight risk to become counterproductive. Thus estimates older than a given threshold must be deleted or updated. In order to perform this, we use the timestamp reference: when it becomes older than a threshold then we send a new load estimate. These estimates can be analyzed in order to calculate the distribution of the load among the nodes and the number of nodes in the network.

3.2. 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 [9]. 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 according to the alphabetic order, C flushes each data whose hash is greater than the ID assigned to the new node.

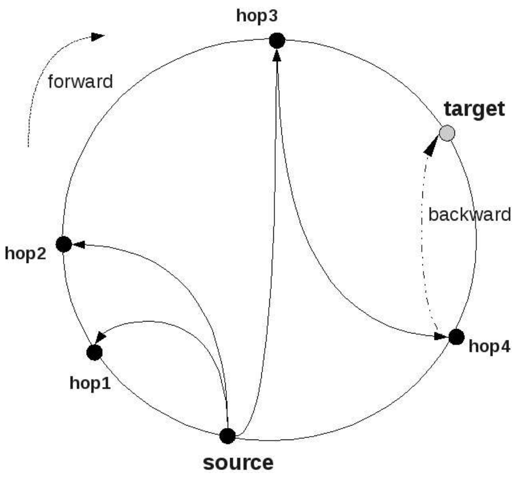

Figure 3. Routing loop management.

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 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 [12] 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 a temporary node unavailability.

3.3. Routing messages

Following the idea proposed in [2] each node has a DHT composed of two siblings 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 $O(\frac{log^2(n)}{k})$ [11], where k is the number of long distance links [10], that is $log(n)$ as shown in [2].

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 $p_n(x) = \frac{1}{n*log(x)}$, if $x \in [\frac{1}{n}, 1]$, 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 in-

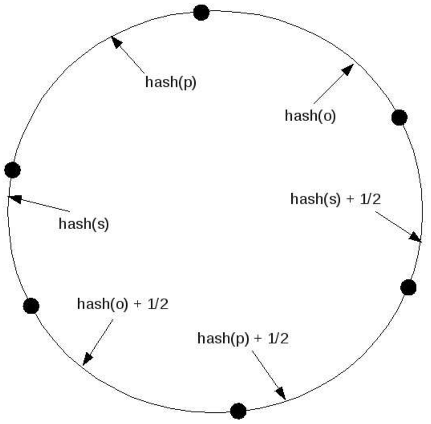

Figure 4. Redundant copies of a given triple (s,p,o) with t = 2.

consistent 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 $O(n)$.

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:

$$\sum_{i=0}^{i=t-1} hash(x) + \frac{\omega * i}{t}, \qquad (1)$$

where x is a URI field of a statement, ω is equal to 2^{hash_length} and t identifies the number of replicas. Fig. 4 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 (1) 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 the same time independently from the particular node used to perform the query.

We identify three types of queries: atomic, disjunctive and conjunctive queries. 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 uses the hash function on the query pattern exact values to identify the node which manages the requested triple-set. The routing cost follows the trend described in Section 3.3.

A disjunctive query pattern is an atomic query with a list of constraints. It is then evaluated as an atomic query but the node responsible for such triple-set has to return only the data that matches the specified constraints.

A conjunctive query pattern 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, described in [7]. 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.

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. 5 and Fig. 6 we represent the relative standard deviation (RSD) of the number of triples managed by each peer in two simulation scenarios.

In Fig. 5 300 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

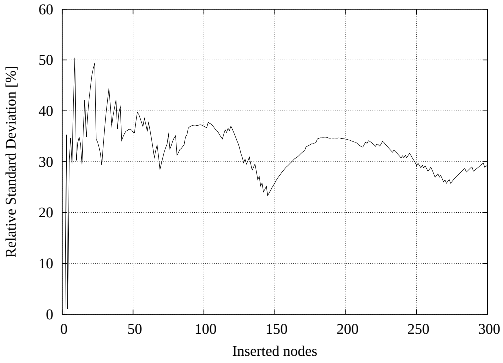

Figure 5. Load balancing in the static scenario.

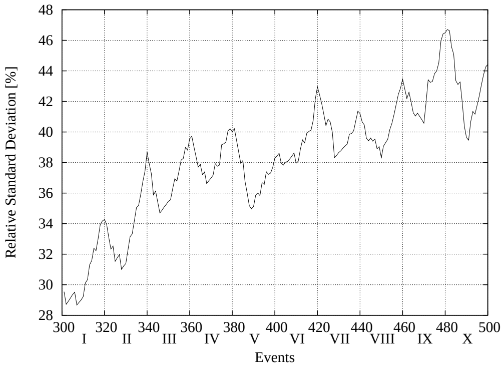

Figure 6. Load balancing in the dynamic scenario.

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. 6, 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. 6, the bandwidth saving load balancing algorithm applied to the insertion events can only slowly reduce the RSD.

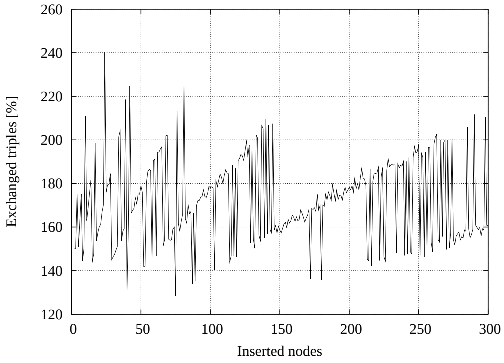

Figure 7. Network bandwidth usage in the static scenario.

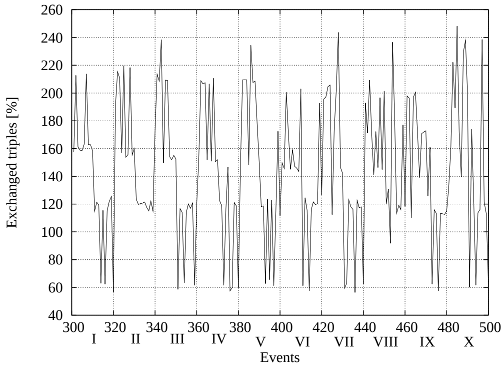

Figure 8. Network bandwidth usage in the dynamic scenario.

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. 7 and 8. The number of triples exchanged are 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.

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.
We also plan to optimize the query resolving algorithm with respect to the network bandwidth and node computational load using statistical information on attribute selectivity. Since some URI, e.g. rdf:type or dc:title, occur more frequently than others, they are also the least selective one. Thus we are working on the implementation of query chains that analyze query constraints depending on their selectivity in order to reduce the set of triples to be evaluated with respect to a query chain that uses a random evaluation order.
Besides we will also investigate how an 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.

6. Acknowledgements

We wish to thank Andrea Benazzo for his initial studies on building the RDF infrastructure upon a set of P2P protocols. This work was partially supported by the Regione Piemonte through the VICSUM project.

References

[1] E. P. Andy Seaborne. SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/, January 2008.

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

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

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

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

[7] M. K. Erietta Liarou, Stratos Idreos. Evaluating conjunctive triple pattern queries over large structured overlay networks. In *International Semantic Web Conference*, pages 399–413, 2006.

[8] M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica. 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] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. 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.

[10] J. Kleinberg. 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.

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

[12] S. Milgram. *The small world problem*. Psychology Today, May 1967.

[13] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. 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.

[14] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. 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.

[15] H. S. Steffen Staab. *Semantic Web and Peer-to-Peer*. Springer, 2006.

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